

Java e Serviço de Transporte

Vitor Brandi Junior

Sumário

- Introdução
- O pacote java.net
- A classe InetAddress
- A classe URL
- A classe URLConnection
- Comunicação Sem Conexão
 - definições
 - exemplo
- Comunicação Com Conexão
 - definições
 - exemplo
- Bibliografia

Introdução

- Sistemas distribuídos são compostos por vários programas executados em diferentes processadores
- A comunicação entre estes processadores pode ser fortemente ou fracamente acoplada
 - **Fortemente acoplada:** a comunicação ocorre através de memória compartilhada
 - **Fracamente acoplada:** a comunicação ocorre através de uma rede de comunicação

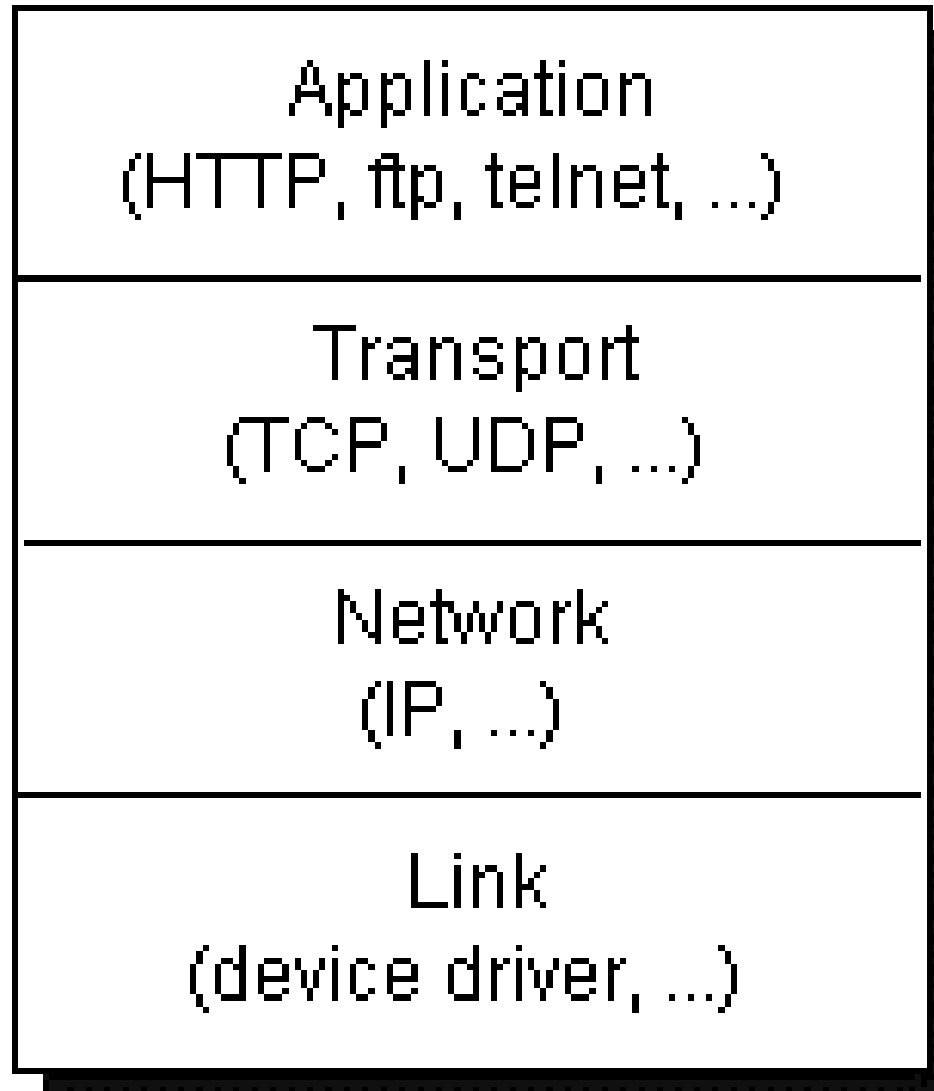
Introdução

- Em redes TCP/IP o acesso aos serviços providos pela rede ocorre através de APIs, tais como:
 - TLI (*Transport Layer Interface*)
 - Sockets
- Em uma rede TCP/IP a comunicação ocorre através de serviços oferecidos pelos protocolos de transporte TCP e UDP

Introdução

- Os serviços oferecidos por estes protocolos de transporte incluem:
 - criar pontos de transporte
 - alocar recursos para comunicação
 - estabelecer conexões
 - aguardar conexões
 - enviar dados
 - receber dados
 - terminar ou abortar conexões
 - tratar erros

Esquematicamente



O pacote `java.net.*`

- Quando se escreve programas que se comunicam através de uma rede programa-se no nível de aplicação
- Para se fazer uso dos serviços de transporte deve-se utilizar as classes do pacote `java.net`
- Estas classes provém implementações de serviços de transporte que são independentes de plataforma

O pacote `java.net.*`

- As principais classes do pacote `java.net` que implementam serviços TCP são:
 - `URL`
 - `URLConnection`
 - `Socket`
 - `ServerSocket`
- Já as classes que implementam serviços UDP são:
 - `DatagramPacket`
 - `DatagramSocket`
 - `MulticastSocket`

A classe InetAddress

- Em uma rede TCP/IP as máquinas na rede são identificadas por endereços com tamanho fixo denominados IP
- A classe `java.net.InetAddress` armazena o endereço IP uma máquina
- Seus principais métodos são:
 - `byte[] getAddress()`: retorna o endereço IP do objeto `InetAddress`
 - `static InetAddress[] getAllByName(String host)`: retorna todos os endereços de IP de um computador a partir do nome deste computador

A classe InetAddress

- static InetAddress
getByName(String host): determina o endereço de IP de um computador a partir de seu nome
- String getAddress(): retorna o endereço de IP de um computador
- String getHostName(): retorna o nome do computador identificado pelo IP
- static InetAddress
getLocalHost(): retorna o endereço do computador local

Exemplos de InetAddress

```
import java.net.*;

class ControladorAcesso
{
    public static void main (String args[])
    {
        InetAddress enderecoIP = null;
        try
        {
            enderecoIP = InetAddress.getLocalHost();
            System.out.println("Endereco: " + enderecoIP.getHostAddress());
        }
        catch(UnknownHostException e)
        {
            System.out.println("Erro: " + e.getMessage());
        }
    }
}
```

Explicando o exemplo:

- O método `getLocalHost()` retorna uma instância da classe `InetAddress` que descreve o endereço do computador local
- A representação do endereço através de uma cadeia de caracteres é feita através do método `getHostAddress()`

Outro exemplo:

```
import java.net.*;
import java.io.*;

class ControladorAcesso1
{
    public static void main (String args[])
    {
        InetAddress enderecoIP = null;
        String nome;
        nome = args[0];
        try
        {
            enderecoIP = InetAddress.getByName(nome);
            System.out.println("Endereco: " + enderecoIP.getHostAddress());
        }
        catch(UnknownHostException e)
        {
            System.out.println("Erro: " + e.getMessage());
        }
    }
}
```

Explicando outro exemplo:

- Neste exemplo o nome do servidor é passado como parâmetro para o programa
- o método `getByName ()` retorna uma instância de `InetAddress` obtida a partir do nome do servidor
- o método `getHostAddress ()` devolve o endereço de IP do objeto `InetAddress`
- **Ex:** `java ControladorAcesso1 localhost`
`Endereco: 127.0.0.1`

Mais outro exemplo:

```
import java.net.*;
import java.io.*;

class ControladorAcesso2
{
    public static void main (String args[])
    {
        InetAddress[] enderecosIP = null;
        String nome;
        nome = args[0];
        try
        {
            enderecosIP = InetAddress.getAllByName(nome);
            for (int i = 0; i < enderecosIP.length; i++)
                System.out.println("Endereco: " + enderecosIP[i].getHostAddress())
        }
        catch(UnknownHostException e)
        {
            System.out.println("Erro: " + e.getMessage());
        }
    }
}
```

Explicando...

- No caso de uma máquina possuir mais de um endereço de IP (um roteador, p.ex.), o método `getAllByName()` retorna uma instância de `InetAddress` para cada endereço IP da máquina
- **Ex:**

```
java ControladorAcesso2 localhost  
Endereco: 127.0.0.1
```


Ainda outro exemplo:

```
import java.net.*;
import java.io.*;

class ControladorAcesso3
{
    public static void main (String args[])
    {
        InetAddress enderecoIP = null;
        String nome;
        nome = args[0];
        try
        {
            enderecoIP = InetAddress.getByName(nome);
            System.out.println("Endereco: " + enderecoIP.getHostAddress());
            System.out.println("Nome....: " + enderecoIP.getHostName());
        }
        catch(UnknownHostException e)
        {
            System.out.println("Erro: " + e.getMessage());
        }
    }
}
```

Explicando...

- O método `getHostName()` possibilita que seja obtido o nome associado à uma máquina cujo endereço é identificado por uma instância de `InetAddress`.
- **Ex:**

```
java ControladorAcesso3 localhost
Endereco: 127.0.0.1
Nome....: localhost
```

URL

- Todo recurso da World-Wide Web tem um endereço que pode ser codificado como um URL (Uniform Resource Locator)
- A classe `URL` do pacote `java.net` representa o endereço segundo esta codificação.
- Seus principais métodos são:
 - `getContent()`: retorna o conteúdo do URL
 - `getFile()`: retorna o nome do arquivo

URL

- Principais métodos (cont):
 - `getHost()`: retorna o nome da máquina
 - `getPort()`: retorna o número da porta
 - `getProtocol()`: retorna o nome do protocolo
 - `getRef()`: retorna a âncora (`#dest`)
 - `openConnection()`: abre uma conexão e retorna um `URLConnection`
 - `openStream()`: abre uma conexão e retorna um `InputStream`
 - `sameFile()`: compara dois URLs
 - `set()`: especifica os campos do URL

URL

- Principais métodos (cont):
 - `toExternalForm()` : representa o URL como uma cadeia de caracteres
- Construtores:
 - `public URL (String protocolo, String host, int porta, String arquivo)`
 - `public URL (String protocolo, String host, String arquivo)`
 - `public URL (String nome)`

Exemplo de URL

```
import java.net.*;
import java.io.*;
class ControladorAcesso4
{
    public static void main (String args[])
    {
        URL url = null;
        String nome;
        nome = args[0];
        try
        {
            url = new URL(nome);
            System.out.println("Protocolo: " + url.getProtocol());
            System.out.println("Maquina..: " + url.getHost());
            System.out.println("Porta....: " + url.getPort());
            System.out.println("Arquivo...: " + url.getFile());
            System.out.println("Ancora...: " + url.getRef());
            System.out.println("Formatado: " + url.toExternalForm());
        }
        catch(MalformedURLException e)
        {
            System.out.println("Erro: " + e.getMessage());
        }
    }
}
```

Executando o exemplo:

- **Ex:** `java ControladorAcesso4 http://localhost:8080/login.html`
Protocolo: http
Maquina...: localhost
Porta....: 8080
Arquivo...: /login.html
Ancora...: null
Formatado: http://localhost:8080/login.html

URLConnection

- O método `openConnection()` retorna uma referência para uma instância da classe `java.net.URLConnection`
- Esta classe descreve uma conexão entre o programa e o recurso identificado pelo URL
- Seus principais métodos são:
 - `connect()` : estabelece a conexão
 - `getContent()` : recupera o conteúdo da conexão

URLConnection

- Principais métodos (cont):
 - `getContentEncoding()` : retorna o valor do campo *content-encoding*
 - `getContentLength()` : retorna o valor do campo *content-length*
 - `getContentType()` : retorna o valor do campo *content-type*
 - `getDate()` : retorna o valor do campo *date*
 - `getExpiration()` : retorna o campo *expires*
 - `getHeaderField()` : retorna o valor do campo de cabeçalho

URLConnection

- Principais métodos (cont):
 - `getInputStream()` : retorna stream para leitura
 - `getLastModified()` : retorna o valor do campo *last-modified*
 - `getOutputStream()` : retorna stream para escrita
 - `getURL()` : retorna o URL da conexão

Exemplo de URLConnection

```
import java.net.*;
import java.io.*;
class ControladorAcesso5 {
    public static void main (String args[]) {
        URL url = null;
        URLConnection conexaoURL = null;
        String nome;
        nome = args[0];
        try {
            url = new URL(nome);
            conexaoURL = url.openConnection();
            System.out.println("content-length: : " + conexaoURL.getContentLength());
            System.out.println("content-type....: " + conexaoURL.getContentType());
            System.out.println("expiration.....: " + conexaoURL.getExpiration());
            System.out.println("date.....: " +
                               new java.util.Date(conexaoURL.getDate()).toLocaleString());
        }
        catch(MalformedURLException e) {
            System.out.println("Erro: " + e.getMessage());
        }
        catch(IOException i) {
            System.out.println("Erro: " + i.getMessage());
        }
    }
}
```

Executando o exemplo:

- **Ex:**

```
java ControladorAcesso5 http://localhost:8080/login.html
content-length..: 562
content-type....: text/html
expiration.....: 1005631750000
date.....: Tue Nov 13 04:09:05 GMT-02:00 2001
```

Outro exemplo de URLConnection

```
import java.net.*;
import java.io.*;
class ControladorAcesso6 {
    public static void main (String args[]) {
        URL url = null;
        URLConnection conexaoURL = null;
        String nome;
        String linha;
        BufferedReader bufferedReader;
        nome = args[0];
        try {
            url = new URL(nome);
            conexaoURL = url.openConnection();
            bufferedReader = new BufferedReader(
                new InputStreamReader(conexaoURL.getInputStream()));
            while((linha = bufferedReader.readLine())!=null)
                System.out.println(linha);
        }
        catch(MalformedURLException e) {
            System.out.println("Erro: " + e.getMessage());
        }
        catch(IOException i){
            System.out.println("Erro: " + i.getMessage());
        }
    }
}
```

Executando o exemplo:

```
C:\> java ControladorAcesso5
      http://localhost:8080/teste.html
<title>404 Not Found</title>
<h1>404 Not Found</h1>
/teste.html was not found on this server.<p>
<hr>
<small>
Resin 1.1.3 -- Thu Jun 29 12:04:45 PDT 2000
</small>
</address>
```

Relembrando...

- A Arquitetura Cliente-Servidor
 - um cliente envia mensagens solicitando serviços e aguarda mensagens com respostas
 - um servidor aguarda mensagens solicitando serviços, presta serviços e devolve mensagens com respostas
 - os serviços podem ser prestados por um único thread (atendimento sequencial) ou por vários threads (atendimento concorrente)

Pontos de transporte

- Para que clientes e servidores possam trocar dados através da rede, é necessário que sejam criados pontos de transporte (*transport endpoints*) e que endereços sejam a eles associados.
- O pacote `java.net` traz as seguintes classes para a definição destes pontos:
 - `Socket`: ponto de transporte de um cliente
 - `ServerSocket`: ponto de transporte de um servidor
 - `DatagramSocket`: ponto de transporte não orientado a a conexão

Comunicação sem Conexão

- É realizada através dos serviços providos pelo protocolo UDP
- Os serviços do UDP não garantem a entrega dos dados enviados
- A responsabilidade por identificar a perda de dados e retransmitir estes dados é do próprio programa

Comunicação sem Conexão

- Para se comunicar, um cliente não orientado a conexão deve executar os seguintes passos:
 - identificar o endereço da máquina onde o serviço é prestado
 - identificar o número da porta onde o serviço é prestado
 - criar um ponto de transporte
 - enviar e receber dados através deste ponto de transporte
 - liberar o ponto de transporte quando não mais necessário

Comunicação sem Conexão

- Para se comunicar, um servidor não orientado a conexão deve executar os seguintes passos:
 - criar um ponto de transporte
 - associá-lo ao número da porta onde o serviço é prestado
 - receber solicitação de serviço
 - executar o serviço
 - enviar a resposta
 - voltar a aguardar a solicitação de serviço

Comunicação sem Conexão

- Em Java a comunicação não orientada a conexão utiliza as classes `DatagramPacket` e `DatagramSocket`
- **DatagramPacket**
 - seus principais métodos são:
 - `getAddress()`: retorna o endereço associado ao pacote
 - `getData()`: retorna os dados associados ao pacote
 - `getLength()`: retorna o tamanho do pacote
 - `getPort()`: retorna a porta associada ao pacote

Comunicação sem Conexão

- **DatagramSocket**
 - seus principais métodos são:
 - `close()`: fecha o ponto de transporte
 - `getLocalPort()`: retorna a porta associada ao ponto de transporte
 - `receive()`: recebe o pacote
 - `send()`: envia o pacote
 - `getSoTimeout()`: retorna o tempo máximo de espera
 - `setSoTimeout()`: especifica o tempo máximo de espera

Comunicação sem Conexão

- Para enviar e receber dados é necessário criar-se uma instância de cada uma destas classes
- Feito isto são utilizados os métodos `send()` e `receive()` da classe `DatagramSocket` para envio e recepção de dados
- Para envio dos dados são necessárias instâncias de `InetAddress` e `DatagramPacket`
- A instância de `DatagramPacket` armazena informações sobre os dados a serem enviados (referência para o array de dados a serem enviados, quantidade de bytes, endereço do servidor, porta no servidor)

Comunicação sem Conexão

- Exemplo no cliente:

```
byte[] dados;  
int portaServidor;  
InetAddress enderecoServidor;  
DatagramPacket pacoteEnviar;  
DatagramSocket datagramSocket;  
...  
pacoteEnviar = new DatagramPacket(dados, dados.length,  
                                   enderecoServidor, portaServidor);  
  
...  
try {  
    datagramSocket = new DatagramSocket();  
}  
catch(SocketException s) {  
}  
...  
try {  
    datagramSocket.send(pacoteEnviar);  
}  
catch(IOException i) {  
}
```

Comunicação sem Conexão

- Como não há garantia de que os pacotes são entregues, o cliente deve:
 - aguardar pela resposta do servidor
 - se esta resposta não vier, deve transmitir novamente a solicitação de serviço
 - se isto se repetir um certo número de vezes, o cliente deve considerar que há um problema e tomar providências adequadas
 - este comportamento pode ser obtido através do método `setSoTimeout()`, que especifica o tempo máximo (em milésimos de segundos) que o cliente fica bloqueado aguardando por um pacote após invocado o método `receive()`
 - estourado este tempo, é lançada uma exceção

Comunicação sem Conexão

- Exemplo no cliente:

```
try
{
    datagramSocket = new DatagramSocket();
    datagramSocket.setSoTimeout(TEMPO_ESPERA_MAXIMO);
}
catch(SocketException s)
{
    System.out.println(s.getMessage());
    return;
}
```

Comunicação sem Conexão

- Na recepção dos dados utiliza-se uma instância de `DatagramPacket` que informa a referência para a área de memória onde os dados recebidos serão armazenados, além também do tamanho desta área
- Se o código faz parte de um servidor, ao se instanciar a classe `DatagramSocket` é necessário especificar o número da porta através da qual o serviço é prestado

Comunicação sem Conexão

- Exemplo no servidor:

```
byte[] dados;  
DatagramPacket pacote;  
DatagramSocket datagramSocket;  
.....  
try {  
    datagramSocket = new DatagramSocket(PORTA_SERVICO);  
}  
catch(SocketException s) {  
}  
.....  
dados = new byte[TAMANHO_MAXIMO];  
pacote = new DatagramPacket(dados, dados.length);  
try {  
    datagramSocket.receive(pacote);  
}  
catch(IOException i) {  
}
```

Exemplo de um cliente UDP

```
import java.net.*;
import java.io.*;

class ClienteDataHorarioUDP
{
    private static final int TAMANHO_MAXIMO = 100;
    private static final int PORTA_SERVICO = 2000;
    private static final int TEMPO_ESPERA_MAXIMO = 500;
    private static final int NUMERO_MAXIMO_TENTATIVAS = 10;

    public static void main(String args[])
    {
        String nomeServidor;
        String operacaoSolicitada;
        byte[] dados;
        int portaServidor = PORTA_SERVICO;
        int numeroTentativas;
        boolean enviarSolicitacao;
        InetAddress enderecoServidor = null;
        DatagramPacket pacoteEnviar = null;
        DatagramPacket pacoteReceber = null;
        DatagramSocket datagramSocket = null;
        nomeServidor = args[0];
        operacaoSolicitada = args[1];
    }
}
```

Exemplo de um cliente UDP

```
try {
    enderecoServidor = InetAddress.getByName(nomeServidor);
}
catch(UnknownHostException u) {
    System.out.println(u.getMessage());
    return;
}
switch (new Integer(operacaoSolicitada).intValue()) {
    case 1:
        operacaoSolicitada = "DATA";
        break;
    case 2:
        operacaoSolicitada = "HORA";
        break;
    default:
        System.out.println("Operacao invalida");
        return;
}
dados = operacaoSolicitada.getBytes();
pacoteEnviar = new DatagramPacket(dados,dados.length,enderecoServidor,portaServidor);
try {
    datagramSocket = new DatagramSocket();
    datagramSocket.setSoTimeout(TEMPO_ESPERA_MAXIMO);
}
catch(SocketException s) {
    System.out.println(s.getMessage());
    return;
}
dados = new byte[TAMANHO_MAXIMO];
pacoteReceber = new DatagramPacket(dados, dados.length);
enviarSolicitacao = true;
numeroTentativas = 0;
```

Exemplo de um cliente UDP

[illegible]

Exemplo de um servidor UDP

```
import java.net.*;
import java.io.*;
import java.util.*;
class ServidorDataHorarioUDP
{
    private static final int TAMANHO_MAXIMO = 100;
    private static final int PORTA_SERVICO = 2000;
    public static void main(String args[])
    {
        String operacaoSolicitada;
        String resposta;
        byte[] dados;
        int portaCliente;
        Date data;
        Calendar calendario;
        InetAddress enderecoCliente = null;
        DatagramPacket pacote = null;
        DatagramSocket datagramSocket = null;
        calendario = new GregorianCalendar();
        try
        {
            datagramSocket = new DatagramSocket(PORTA_SERVICO);
        }
        catch(SocketException s)
        {
            System.out.println(s.getMessage());
            return;
        }
    }
}
```

Exemplo de um servidor UDP

```
while(true)
{
    dados = new byte[TAMANHO_MAXIMO];
    pacote = new DatagramPacket(dados, dados.length);

    try
    {
        datagramSocket.receive(pacote);
    }
    catch(IOException i)
    {
        System.out.println(i.getMessage());
        return;
    }

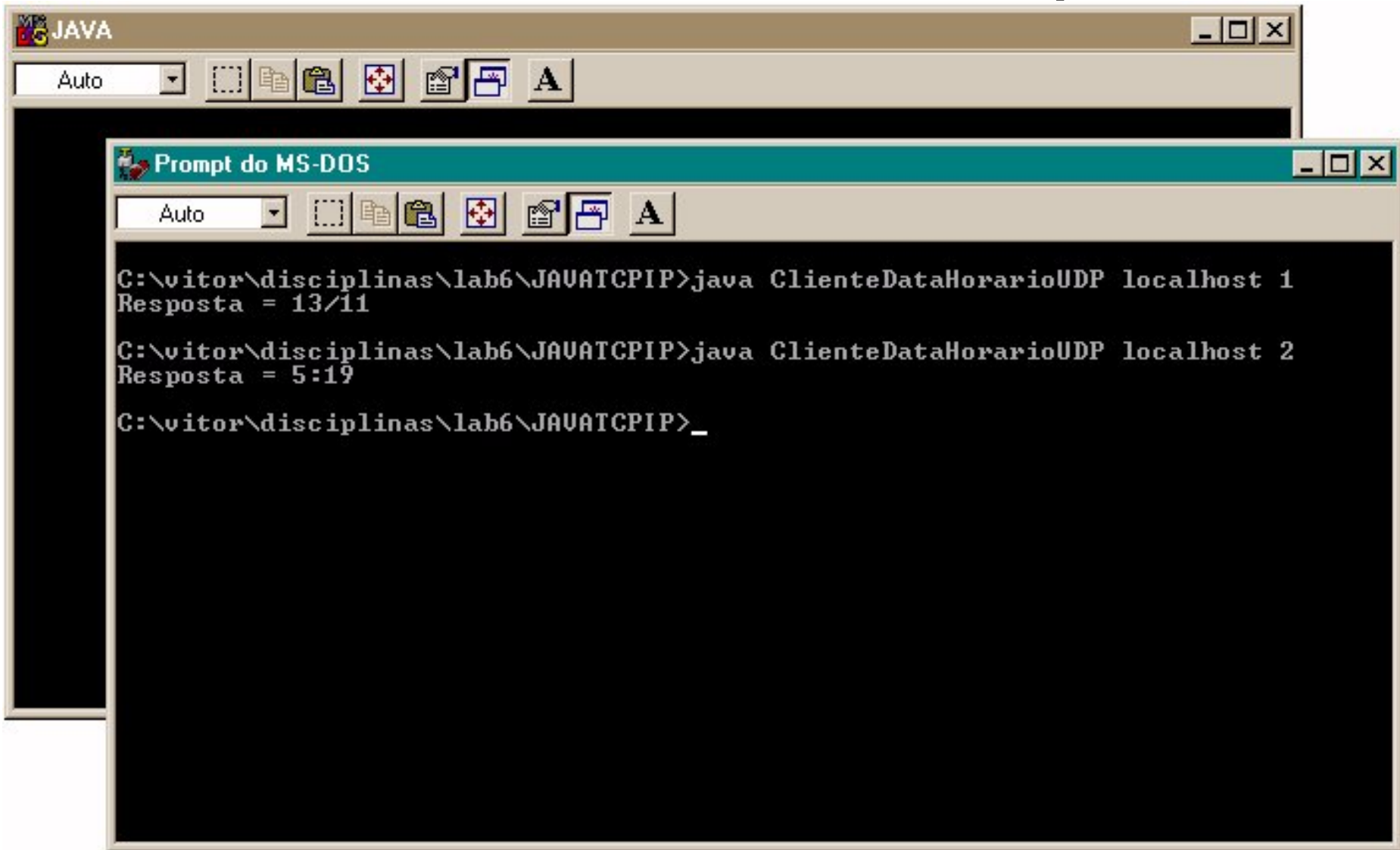
    enderecoCliente = pacote.getAddress();
    portaCliente = pacote.getPort();
    operacaoSolicitada = new String(pacote.getData(), 0, pacote.getLength());

    if (operacaoSolicitada.compareTo("DATA")==0)
    {
        data = new Date();
        calendario.setTime(data);
        resposta = calendario.get(Calendar.DATE)+ "/" +(calendario.get(Calendar.MONTH)+1);
    }
}
```


Exemplo de um servidor UDP

```
else
{
    if (operacaoSolicitada.compareTo("HORA")==0)
    {
        data = new Date();
        calendario.setTime(data);
        resposta = calendario.get(Calendar.HOUR)+":" + calendario.get(Calendar.MINUTE)
    }
    else
    {
        resposta = "Servico nao implementado";
    }
}
dados = resposta.getBytes();
pacote = new DatagramPacket(dados, dados.length, enderecoCliente, portaCliente);
try
{
    datagramSocket.send(pacote);
}
catch(IOException i)
{
    System.out.println(i.getMessage());
    return;
}
}
```

Executando o exemplo



The image shows a screenshot of a computer screen with two windows. The top window is titled 'JAVA' and has a toolbar with icons for 'Auto', 'New', 'Open', 'Save', 'Run', 'Find', and 'Print'. The bottom window is titled 'Prompt do MS-DOS' and has a similar toolbar. The DOS prompt shows the following commands and output:

```
C:\vitor\disciplinas\lab6\JAVATCPIP>java ClienteDataHorarioUDP localhost 1
Resposta = 13/11

C:\vitor\disciplinas\lab6\JAVATCPIP>java ClienteDataHorarioUDP localhost 2
Resposta = 5:19

C:\vitor\disciplinas\lab6\JAVATCPIP>_
```

Comunicação com Conexão

- É efetuada pelos serviços providos pelo protocolo TCP
- Este protocolo responsabiliza-se pela identificação de perda de dados e pela retransmissão destes dados perdidos
- Para tanto os programas precisam gerenciar conexões

Comunicação com Conexão

- Para se comunicar, um cliente orientado a conexão deve:
 - identificar o endereço da máquina onde o serviço é prestado
 - identificar o número da porta onde o serviço é prestado
 - criar um ponto de transporte (Socket)
 - conectar ao servidor
 - enviar e receber dados através do Socket
 - liberar a conexão
 - liberar o ponto de transporte

Comunicação com Conexão

- Para se comunicar, um servidor orientado a conexão deve:
 - criar um ponto de transporte
 - vincular ao ponto de transporte uma porta
 - aguardar a solicitação de conexão
 - aceitar a conexão
 - receber a solicitação de serviço
 - executar o serviço
 - enviar a resposta
 - fechar a conexão
 - voltar a aguardar nova conexão

Comunicação com Conexão

- A comunicação orientada a conexão utiliza-se das classes `Socket` e `ServerSocket`
- Principais métodos de `Socket`
 - `close()` : fecha o ponto de transporte
 - `getInetAddress()` : retorna o endereço ao qual está conectado
 - `getInputStream()` : retorna o *InputStream*
 - `getLocalPort()` : retorna a porta ao qual está conectado
 - `getOutputStream()` : retorna o *OutputStream*
 - `getPort()` : retorna a porta remota ao qual está conectado

Comunicação com Conexão

- Principais métodos da classe

`ServerSocket`:

- `accept()` : aceita a conexão
- `close()` : fecha o ponto de transporte
- `getInetAddress()` : retorna o endereço ao qual está conectado
- `getLocalPort()` : retorna a porta que está sendo escutada

Comunicação com Conexão

- Nos clientes quando `Socket` é instanciado deve-se informar o nome da máquina e o número da porta do servidor com o qual deseja-se estabelecer a conexão
- Feito isto, a conexão é tratada como um stream (fluxo de bytes)
- A partir disto, p. ex., pode-se instanciar `BufferedOutputStream` para se utilizar métodos como `write()` para enviar dados através da conexão
- Semelhantemente, pode-se instanciar `BufferedInputStream` para se utilizar métodos como `read()` para obter dados da conexão

Comunicação com Conexão

- Exemplo no cliente:

```
try {
    socket = new Socket(enderecoServidor, portaServidor);
    bos = new BufferedOutputStream(socket.getOutputStream());
    bis = new BufferedInputStream(socket.getInputStream());
    bos.write(dados, 0, dados.length);
    bos.flush();
}
catch(IOException i) {
}
dados = new byte[TAMANHO_MAXIMO];
try {
    bytesLidos = bis.read(dados, 0, dados.length);
    socket.close();
}
catch(IOException i) {
}
```

Comunicação com Conexão

- Ao se instanciar a classe `ServerSocket` no servidor deve-se especificar o número da porta onde o serviço é prestado
- o método `accept ()` deve ser usado para aceitar pedidos de conexão
- este método retorna uma instância de `Socket`
- É através desta instância que se envia e recebe dados

Comunicação com Conexão

- Exemplo no servidor:

```
ServerSocket serverSocket = null;
Socket socket = null;
try {
    serverSocket = new ServerSocket(PORTA_SERVICO);
    socket = serverSocket.accept();

    bis = new BufferedInputStream(socket.getInputStream());
    dados = new byte[TAMANHO_MAXIMO];
    bytesLidos = bis.read(dados, 0, dados.length);
}
catch(IOException i) {
}
```

Exemplo de um cliente TCP

```
import java.net.*;
import java.io.*;
import java.util.*;
class ClienteDataHorarioTCP {
    private static final int TAMANHO_MAXIMO = 100;
    private static final int PORTA_SERVICO = 2000;

    public static void main(String args[]) {
        String nomeServidor;
        String operacaoSolicitada;
        byte[] dados;
        int bytesLidos;
        int portaServidor = PORTA_SERVICO;
        InetAddress enderecoServidor = null;
        Socket socket = null;
        BufferedOutputStream bos;
        BufferedInputStream bis;
        nomeServidor = args[0];
        operacaoSolicitada = args[1];
        try {
            enderecoServidor = InetAddress.getByName(nomeServidor);
        }
        catch(UnknownHostException u) {
            System.out.println(u.getMessage());
            return;
        }
    }
}
```

Exemplo de um cliente TCP

```
switch (new Integer(operacaoSolicitada).intValue()) {
    case 1:
        operacaoSolicitada = "DATA";
        break;
    case 2:
        operacaoSolicitada = "HORA";
        break;
    default:
        System.out.println("Operacao invalida");
        return;
}

dados = operacaoSolicitada.getBytes();
try {
    socket = new Socket(enderecoServidor, portaServidor);
    bos = new BufferedOutputStream(socket.getOutputStream());
    bis = new BufferedInputStream(socket.getInputStream());
    bos.write(dados, 0, dados.length);
    bos.flush();
}
catch(IOException i) {
    System.out.println(i.getMessage());
    return;
}
```

Exemplo de um cliente TCP

```
        dados = new byte[TAMANHO_MAXIMO];
        try
        {
            bytesLidos = bis.read(dados, 0, dados.length);
            socket.close();
        }
        catch(IOException i)
        {
            System.out.println(i.getMessage());
            return;
        }
        System.out.println("Resposta = " + new String(dados, 0,
                                                         bytesLidos));
    }
}
```

Exemplo de um servidor TCP

```
import java.net.*;
import java.io.*;
import java.util.*;
class ControladorServidorDataHorarioTCP {
    private static final int PORTA_SERVICO = 2000;
    public static void main(String args[]) {
        ServerSocket serverSocket = null;
        Socket socket = null;
        try {
            serverSocket = new ServerSocket(PORTA_SERVICO);
        }
        catch(IOException i){
            System.out.println(i.getMessage());
            return;
        }
        while(true){
            try {
                socket = serverSocket.accept();
            }
            catch(IOException i) {
                System.out.println(i.getMessage());
                return;
            }
            Thread servidorDataHorarioTCP;
            servidorDataHorarioTCP = new ServidorDataHorarioTCP(socket);
            servidorDataHorarioTCP.start();
        }
    }
}
```

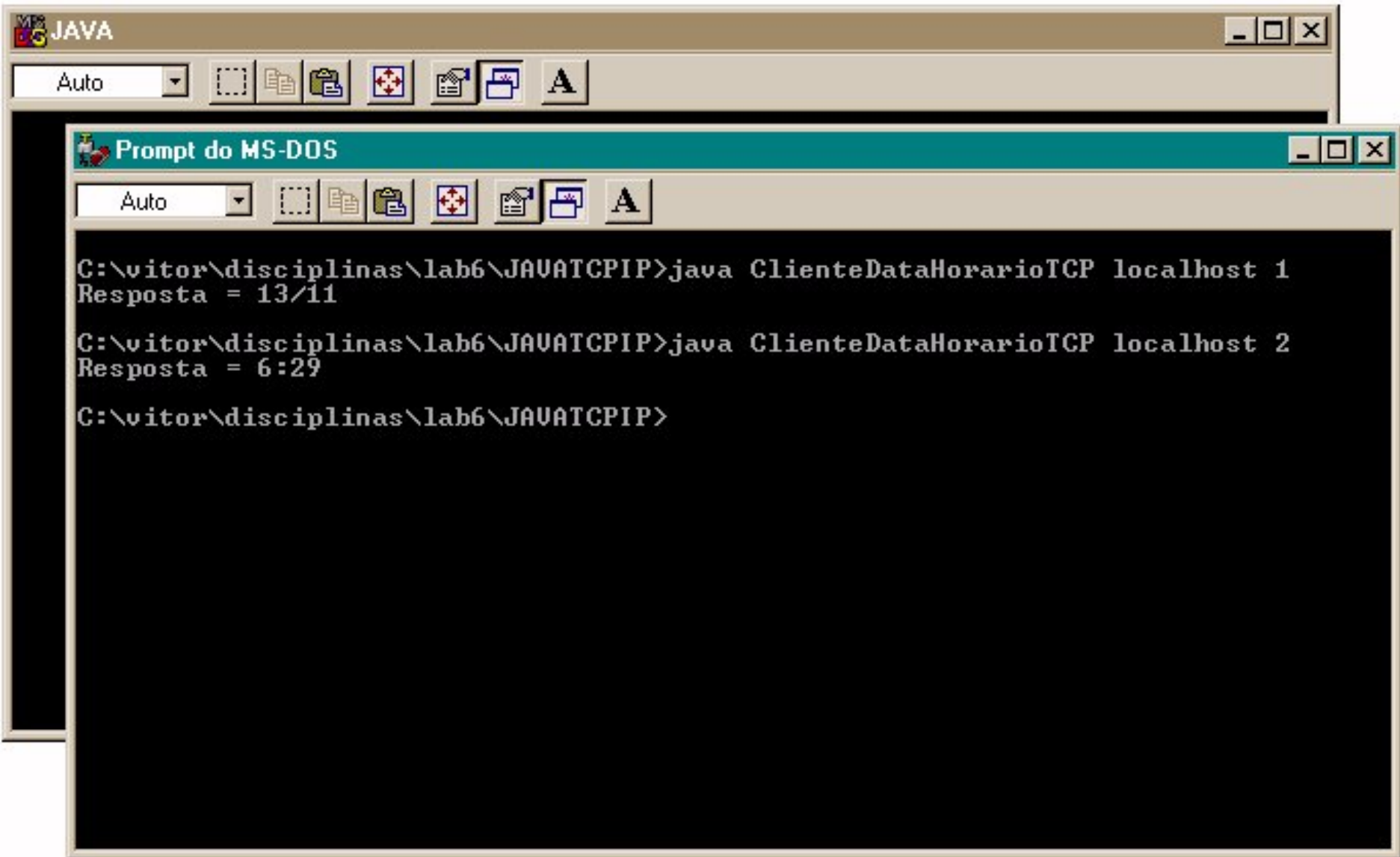
Exemplo de um servidor TCP

```
class ServidorDataHorarioTCP extends Thread {
    private static final int TAMANHO_MAXIMO = 100;
    String operacaoSolicitada;
    String resposta;
    byte[] dados;
    int bytesLidos;
    Date data;
    Calendar calendario;
    Socket socket;
    BufferedOutputStream bos;
    BufferedInputStream bis;
    public ServidorDataHorarioTCP(Socket socket) {
        this.socket = socket;
        calendario = new GregorianCalendar();
    }
    public void run() {
        try {
            bos = new BufferedOutputStream(socket.getOutputStream());
            bis = new BufferedInputStream(socket.getInputStream());
        }
        catch(IOException i){
            System.out.println(i.getMessage());
            return;
        }
    }
}
```


Exemplo de um servidor TCP

```
dados = new byte[TAMANHO_MAXIMO];
try {
    bytesLidos = bis.read(dados, 0, dados.length);
}
catch(IOException i) {
    System.out.println(i.getMessage());
    return;
}
operacaoSolicitada = new String(dados, 0, bytesLidos);
if (operacaoSolicitada.compareTo("DATA")==0) {
    data = new Date();
    calendario.setTime(data);
    resposta=calendario.get(Calendar.DATE)+"/"+(calendario.get(Calendar.MONTH)+1)
}
else {
    if (operacaoSolicitada.compareTo("HORA")==0) {
        data = new Date();
        calendario.setTime(data);
        resposta=calendario.get(Calendar.HOUR)+":"+calendario.get(Calendar.MINUTE);
    }
    else {
        resposta = "Servico nao implementado";
    }
}
dados = resposta.getBytes();
try {
    bos.write(dados, 0, dados.length);
    bos.flush();
}
catch(IOException i){
    System.out.println(i.getMessage());
    return;
```

Executando o exemplo:



The image shows a screenshot of a computer screen with two windows. The top window is titled 'JAVA' and has a toolbar with icons for file operations and a text editor. The bottom window is titled 'Prompt do MS-DOS' and shows a command prompt with the following text:

```
C:\vitor\disciplinas\lab6\JAVATCPIP>java ClienteDataHorarioTCP localhost 1
Resposta = 13/11

C:\vitor\disciplinas\lab6\JAVATCPIP>java ClienteDataHorarioTCP localhost 2
Resposta = 6:29

C:\vitor\disciplinas\lab6\JAVATCPIP>
```

Bibliografia

- Albuquerque, F. “TCP/IP Internet - Programação de Sistemas Distribuídos”, Ed. Axcel Books, 2001
- Sun - “The Java Tutorial”, 2001
- Sun - “Java API Documentation”, 2001