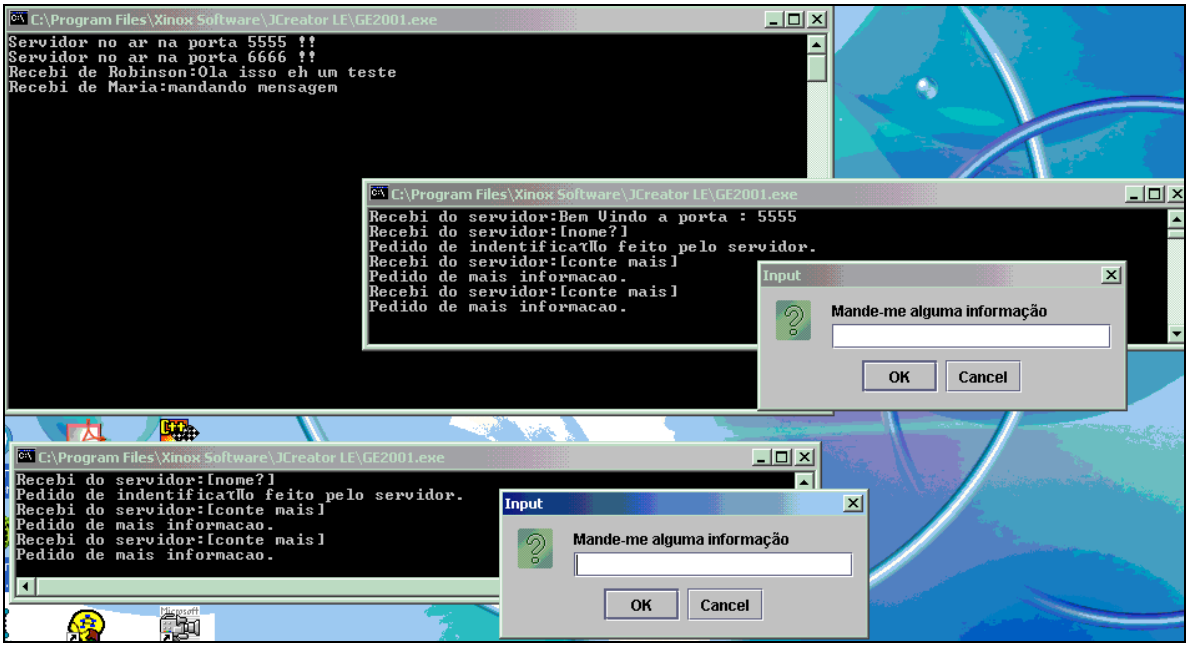
Java : Comunicação Cliente-Servidor.

Objetivo:

- Capacitar o aluno a desenvolver uma aplicação servidora para múltiplos clientes.
- Apresentar as classes Socket e ServerSocket, classes de fluxo de dados, a classe Thread, a interface Runnable e a classe Vector.
- Apresentar, também, a prática de definição de protocolos de comunicação.



Sumário

Introdução

Uma aplicação servidora deve ser capaz de fornecer serviços a clientes que se conectem a ela. Através de uma rede TCP/IP, um servidor gerencia uma porta de entrada. O TCP/IP é um conjunto de protocolos desenvolvido de modo a permitir que computadores compartilhem recursos através da rede. Qualquer cliente que queira acessar os serviços de uma aplicação, o faz através dessa porta de entrada.

O servidor necessita, constantemente, verificar na porta se algum cliente está solicitando um serviço. Essa ação pode ser realizada em Java, através da classe *Thread* ou da interface *Runnable*. *Threads* e *Runnables* são executadas de forma concorrente com o sistema operacional e outras aplicações.

Uma *Thread* ou *Runnable* possuem a assinatura de método `public void run()`. Quando disparada, tanto a *Thread* quanto a *Runnable*, executam esse método de forma concorrente com os outros serviços do sistema operacional.

1. Threads

A classe Thread

A classe *Thread* permite definir um serviço no sistema operacional. A Java Virtual Machine permite que uma aplicação possua diversos serviços sendo executadas de forma concorrente. No exemplo abaixo, a classe Servidor extende essa classe Thread.

Exemplo de Thread

```
/*01*/ ///////// arquivo Servidor.java
/*02*/ public class Servidor extends Thread
/*03*/ {
/*04*/      int i=0;

/*05*/      public void run()
/*06*/      {
/*07*/            while(true)
/*08*/             {
/*09*/              System.out.print("\r"+(++i));
/*10*/             }
/*11*/      }

/*12*/      public static void main(String[] a)
/*13*/      { Servidor s = new Servidor();
/*14*/        s.start(); //disparo da Thread
/*15*/       } }
```

Inicialização da Thread:

A inicialização da *Thread* é feita através da chamada do método `start()`, conforme exemplificado na linha 14 do código anterior. Essa instrução irá executar o serviço definido no método `public void run()`.

Exemplo de chamada de duas Threads:

```
/*01*/ ///////// arquivo Servidor.java
/*02*/ public class Servidor extends Thread {
/*03*/      int i=0;
/*04*/      int espera=0;

/*05*/      public Servidor(String n,int tempo)
/*06*/      {setName(n);espera=tempo;}

/*07*/      public void run()
```

```
/*08*/      {
/*09*/        while(true)
/*10*/      {  System.out.print("\r" + getName()
/*11*/                                +":" + (++i));
/*12*/           try {sleep(espera);}
/*13*/           catch (InterruptedException erro){}
/*14*/        } }

/*15*/      public static void main(String[] a)
/*16*/      { Servidor s = new Servidor("contador",1000);
/*17*/        s.start();
/*18*/        Servidor s2 = new Servidor("cont",100);
/*19*/        s2.start();
/*20*/       } }
```

Comentários:

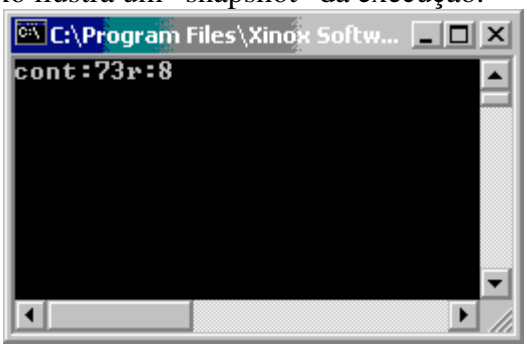
O primeiro exemplo, o arquivo Servidor.java, foi modificado.

Nessa modificação, a função **`sleep( milisegundos )`** é inserida ao código. Essa função permite que a *Thread* adormeça. Esse tempo de espera é fornecido no construtor, bem como um nome para a Thread.

Um destaque deve ser dado à necessidade da estrutura **`try ... catch (InterruptedException)`**.

O método main inicializa dois serviços, `s` e `s2`, conforme exemplificam as linhas 16 a 19 do código anterior.

A figura abaixo ilustra um "snapshot" da execução.



A Thread instanciada por **`s`**, possui o nome **contador** e o tempo de espera de 1000 milisegundos. Cada execução dessa Thread, o nome da thread é impresso, um valor incrementado também é impresso e uma espera de 1000 milisegundos é disparada.

A instância **`s2`**, por outro lado, possui o tempo de espera de 100 milisegundos e o seu nome é **cont**. Ela imprime o seu nome e outro valor é incrementado e impresso.

Observe que o caracter "\r" na instrução `System.out.print`, faz com que essas impressões sejam sobrepostas.

Logo, a thread **`s2`** é executado mais vezes que a thread **`s`**.

2. As Classes Socket e ServerSocket

O que é um Socket?

Um Socket é uma combinação de um computador com o número da porta. Essa combinação identifica uma "porta" de entrada para uma aplicação.

A classe Socket:

Classe que implementa um cliente socket. Um socket é o ponto final da comunicação entre duas máquinas.

A classe ServerSocket:

Classe que implementa um servidor de socket. Um servidor de sockets espera pela solicitação de usuários da rede de computador.. Ele executa algumas operaçÕes baseadas na solicitação recebida e retorna valores.

Dos métodos disponíveis, o principal é o construtor. Nele é definido a porta de leitura. Linhas 8 e 20 do código abaixo.

Exemplo de Servidor com ServerSocket e Socket:

```
/*1*/ ////////////////////// arquivo  Servidor.java
/*2*/  import java.io.*;
/*3*/  import java.net.*;
/*4*/  import java.util.*;

/*5*/  public class Servidor extends Thread
/*6*/  {
/*7*/       private int porta;
/*8*/       private ServerSocket ouvido;
/*9*/       private ObjectOutputStream output;
                     // fluxo de saida de dados.
/*10*/      private ObjectInputStream  inStream;
                    // fluxo de entrada de dados.
/*11*/      private Vector listaJogador = new Vector();
/*12*/      private Jogador visitante;
/*13*/
/*14*/      public void conecta(int porta)
/*15*/      {
/*16*/        // armazena o numero da porta em uma
            // variavel denomiada por "porta"
/*17*/        this.porta = porta;
/*18*/        try
/*19*/        {
/*20*/         ouvido = new java.net.ServerSocket(porta);
              // tenta se conectar com a porta.
/*21*/        }
/*22*/        catch(java.io.IOException e)
              {e.printStackTrace(); System.exit(1);}
   // se deu erro, entao encerre a aplicação.
/*23*/
/*24*/        System.out.println(
             "Servidor no ar na porta " + porta + " !!");
    // se não deu erro, informe que o servidor está no ar
    // na porta definida.
/*25*/      }
/*26*/
/*27*/      public Servidor(String n,int gate)
/*28*/      {setName(n);
/*29*/       conecta(gate);
/*30*/      }
/*31*/
/*32*/      public void run()
/*33*/      {
```

```
/*34*/       while(true)
/*35*/       {
/*36*/        try
/*37*/        {
/*38*/        // servidor.accept() => espera por um
              //cliente , quando o cliente solicita
              //conexao,
/*39*/        // uma instancia da classe Jogador eh
              // criada para representar essa conexao.
/*40*/
/*41*/         visitante =new Jogador(ouvido.accept());
/*42*/         visitante.start();
/*43*/         addJogador(visitante);
/*44*/         visitante.informe(
                 "Bem Vindo a porta : "+porta);
/*45*/        visitante.informe("[nome?]");
/*46*/        } catch (java.io.IOException e) {}
/*47*/       }
/*48*/
/*49*/      }
/*50*/
/*51*/      public static void main(String[] a)
/*52*/      { Servidor s = new
                 Servidor("contador",5555);
/*53*/        s.start();
/*54*/        Servidor s2 = new
                 Servidor("cont",6666);
/*55*/        s2.start();
/*56*/       }
/*57*/
/*58*/      private void removeJogador(Jogador j)
/*59*/      {
/*60*/           listaJogador.remove(j);
/*61*/      }
/*62*/
/*63*/  // adiciona um jogador na lista de jogadores.
/*64*/      private void addJogador (Jogador j)
/*65*/      {
/*66*/        listaJogador.add(j);
/*67*/      }
/*68*/
/*69*/  //////////////definiçao interna de classe
/*70*/  class Jogador extends Thread{
/*71*/      private ObjectOutputStream output;
                    // fluxo de saida de dados.
/*72*/      private ObjectInputStream  inStream;
                   // fluxo de entrada de dados.
/*73*/      private Socket connection;
                  // armazena ponteiro para o servidor.
/*74*/      private boolean noAr=true;
/*75*/      private String nome="";
/*76*/
/*77*/    // construtor da classe
/*78*/      public Jogador(Socket s)
/*79*/      {
/*80*/       connection=s;
                   // armazena a conexao com o servidor.
/*81*/       try{
/*82*/         output = new
                   ObjectOutputStream(s.getOutputStream());
           // cria fluxo de saida de dados.
```

```
/*83*/        inStream = new
                ObjectInputStream(s.getInputStream());
         // cria fluxo de entrada de dados.
/*84*/          output.flush(); // limpa o Buffer.
/*85*/         }
/*86*/       catch (IOException e) {}
/*87*/     }
/*88*/
/*89*/     public void run()
/*90*/     {  String fofoca;
/*91*/        int x;
/*92*/        while(noAr)
/*93*/        {
/*94*/         fofoca=ouvir(false);
/*95*/         x = fofoca.indexOf("bye");
   // se o cliente enviou a instrucao para desconectar-se
   // do servidor, x armazenarah um valor superior a
   // (-1).
/*96*/     if ( x >= 0)
/*97*/        {this.noAr=false;this.interrupt();
/*98*/         Servidor.this.removeJogador(this);
/*99*/         System.out.println(
                   "Pedido de exclusão recebido!");
/*100*/       informe("[exit]");
         // informa ao cliente que ele foi desconectado
/*101*/       continue;
/*102*/      }
/*103*/    if ( fofoca.indexOf("[nome?]") > -1)
/*104*/    {nome = fofoca.substring(7);}
/*105*/    if ( fofoca.indexOf("[conte mais]") > -1)
/*106*/    {System.out.println("Recebi de " + nome + ":"
            + fofoca.substring(12)); }
/*107*/    informe("[conte mais]");
/*108*/    }
/*109*/    }
/*110*/ // foram definidos os seguintes protocolos de
         // comunicacao: [nome?], [conte mais], [exit].

/*111*/ // envia mensagens aos jogadores clientes.
/*112*/     private void informe(String m)
/*113*/     {
/*114*/       try
/*115*/        {
/*116*/          output.writeObject(m);
/*117*/          output.flush();
/*118*/         }
/*119*/          catch (java.io.IOException e)
/*120*/          {
/*121*/                System.out.println(
                           "Erro de envio de mensagem");
/*122*/                this.closeConnection();
/*123*/                noAr=false;
          // se um erro acontecer, entao desconectar o
          // jogador e desligar a Thread. A Thread eh
          // ligada com a instrucao interrupt().
/*124*/           }
/*125*/     }
/*126*/
/*127*/ // rotina de recebimento de mensagens do
         // servidor
/*128*/     private String ouvir(boolean eco)
```

```
/*129*/      {
/*130*/         String saida = new String("");
/*131*/         try {
/*132*/            saida =
                    String.valueOf(inStream.readObject());
/*133*/            if (eco) System.out.println("ouvi " +
/*134*/                  saida);}
/*135*/            catch (IOException e )
/*136*/             {  saida = "";}
/*137*/            catch (java.lang.ClassNotFoundException
                       clerr)
/*138*/           { System.out.println(clerr);}
/*139*/          catch (NullPointerException err){}

/*140*/           return saida;
/*141*/      }
/*142*/
/*143*/  // rotina que permite retirar o jogador da lista
         // de jogadores e desconecta-lo do servidor.
/*144*/      private void closeConnection()
/*145*/     {
/*146*/     try
/*147*/      {Servidor.this.removeJogador(this);
/*148*/       if (this.isAlive())
/*149*/     { this.interrupt();
             // encerra a Thread.
/*150*/     connection.close();}     }
/*151*/     catch (Exception exp)
                {System.out.println("Cliente ausente");}
/*152*/     }
/*153*/    }
/*154*/    // fim da classe interna.
/*155*/  ////////////////////////////
/*156*/  } // fim da classe Servidor
```

Exemplo de Cliente

```
/*1*/ ////////////////// arquivo cliente.java
/*2*/  import java.io.*;
/*3*/  import java.net.*;
/*4*/  import java.util.*;
/*5*/  import javax.swing.*;

/*6*/  class ConversaServidor implements Runnable
/*7*/  {
/*8*/   private Socket con ;
/*9*/   private ObjectOutputStream  output;
/*10*/  private ObjectInputStream   input;
/*11*/  private int p; // porta a ser conectado.
/*12*/
/*13*/  // o construtor cria e abre uma conexao com o
        // servidor.

/*14*/  public ConversaServidor(String endereco
                  , int porta)
          // recebe o endereco e porta como parametros.
/*15*/  {
/*16*/   try
/*17*/   {
/*18*/    con = new
       Socket(InetAddress.getByName(endereco),porta );
/*19*/    output = new
            ObjectOutputStream(con.getOutputStream());
/*20*/    InputStream s = con.getInputStream();
/*21*/    input = new ObjectInputStream(s);
/*22*/  }
/*23*/   catch(java.io.IOException er1)
/*24*/   {System.out.println(er1.getMessage());}
/*25*/   }
/*26*/
/*27*/  // metodo que eh executado quando a Thread eh
        // iniciada - metodo start().
/*28*/  public void run()
/*29*/  {
/*30*/   String texto="";
/*31*/   String mens="";
/*32*/   while(true)
/*33*/   {
/*34*/    try
/*35*/    {
/*36*/      mens = (String)input.readObject();
              // li algo do servidor.
/*37*/     if (mens.length() > 1)
/*38*/     {
/*39*/      System.out.println("Recebi do "
              + " servidor:"+mens);
/*40*/      if (mens.lastIndexOf("[nome?]") > -1)
/*41*/      {texto = "Identifique-se, por favor...";
           mens="[nome?]";
/*42*/      System.out.println("Pedido de indentificação"
            +" feito pelo servidor.");
/*43*/      }
/*44*/      if (mens.lastIndexOf("[conte mais]") > -1)
/*45*/      {texto = "Mande-me alguma informação";
            mens="[conte mais]";
/*46*/       System.out.println(
                "Pedido de mais informacao.");
/*47*/      }
```

```
/*48*/      if (mens.lastIndexOf("[exit]")>-1)
                  {System.exit(0);}
/*49*/      if (texto.length() > 0)
          //repete o protocolo e envia um texto digitado.
               sendData(mens
                 +JOptionPane.showInputDialog(texto));
/*50*/      mens="";texto=""; //limpa tudo
/*51*/       }
/*52*/    }
/*53*/   catch (java.io.OptionalDataException e)
                {System.out.println(
                "Erro no desconectar.\n"
                 +e.getMessage());}
/*54*/   catch (java.lang.ClassNotFoundException e)
           {System.out.println("Class not found. \n"+
             e.getMessage());}
/*55*/   catch (java.io.IOException e)
             {System.out.println("Erro de IO. \n"+
               e.getMessage());}
/*56*/    }
/*57*/  }
/*58*/  // envia mensagens ao servidor.
/*59*/  public void sendData(String m)
/*60*/  {
/*61*/   try{ output.writeObject(m);
/*62*/        output.flush(); // limpa o bbuffer.
/*63*/      }
/*64*/   catch (IOException e) {System.out.println(
             "Nao foi possivel enviar os dados.\n"
               + e.getMessage());}
/*65*/         }
/*66*/  }
/*67*/
/*68*/  public class cliente extends ConversaServidor
/*69*/  {
/*70*/      public cliente(int porta)
/*71*/      {super("127.0.0.1",porta);}
/*72*/
/*73*/      public static void main(String[] a)
/*74*/      {
/*75*/            cliente c1 = new cliente(5555);
/*76*/            c1.run();
/*77*/      }
/*78*/  }
```