Sumário

[**Aulas sobre a Classe Servidor de Continhas** 2](#_Toc87504624)

[**AULA 1 – Aula Servidor de Continhas: uma visão funcional** 3](#_Toc87504625)

[**1.1** **Como funciona/vai ser montado o sistema de Arquitetura Cliente/Servidor** 3](#_Toc87504626)

[**1.2. Lista de Arquivos usados nessa videoaula (Aula 1)** 4](#_Toc87504627)

[**1.2.1.Arquivo: Cliente.java** 4](#_Toc87504628)

[**1.2.1.1.Programa Cliente 1 (de cima):** 5](#_Toc87504629)

[**1.2.1.2.Programa Cliente 2 (debaixo):** 5](#_Toc87504630)

[**1.2.2.Arquivo: Servidor.java** 6](#_Toc87504631)

[**1.2.1.3.Na prompt de comando Programa Cliente 3:** 6](#_Toc87504632)

[**AULA 2 – Fazedor de Continhas: Implementação das classes comuns ao Cliente e ao Servidor** 8](#_Toc87504633)

[**2.1. Lista de Arquivos usados nessa videoaula (Aula 2)** 8](#_Toc87504634)

[**2.1.1.Arquivo: Comunicação.java** 8](#_Toc87504635)

[**2.1.2.Arquivo: PedidoDeOperacao.java** 9](#_Toc87504636)

[**2.1.3.Arquivo: PedidoDeResultado.java** 10](#_Toc87504637)

[**2.1.4.Arquivo: Resultado.java** 10](#_Toc87504638)

[**2.1.5.Arquivo: PedidoParaSair.java** 11](#_Toc87504639)

[**2.1.6.Arquivo: ComunicadoDesDesligamento.java** 11](#_Toc87504640)

[**2.1.7.Arquivo: Teclado.java** 11](#_Toc87504641)

[**2.1.8.Arquivo: Parceiro.java** 12](#_Toc87504642)

[**AULA 3 – Fazedor de Continhas: Implementação das classes específicas de CLIENTE (parte1)** 16](#_Toc87504643)

[**3.1. Como funciona/vai ser montada a classe Cliente** 16](#_Toc87504644)

[**AULA 4 – Fazedor de Continhas: implementação das classes específicas do CLIENTE (parte 2)** 19](#_Toc87504645)

[**4. Lista de Arquivos usados na videoaula (aula 4)** 19](#_Toc87504646)

[**4.1.Arquivo: Cliente.java** 19](#_Toc87504647)

[**4.2.Prompt servidor:** 24](#_Toc87504648)

[**4.3.Prompt Cliente:** 24](#_Toc87504649)

[**4.4.Prompt Servidor:** 25](#_Toc87504650)

[**4.5.No prompt servidor:** 25](#_Toc87504651)

[**4.6.Prompt Cliente:** 25](#_Toc87504652)

[**4.7.Prompt Cliente** 25](#_Toc87504653)

[**4.8.No prompt Cliente:** 26](#_Toc87504654)

[**4.9.Prompt Servidor** 26](#_Toc87504655)

[**4.10.Prompt Cliente** 26](#_Toc87504656)

[**AULA 5 – Fazedor de Continhas: Implementação das classes específicas do servidor (parte 1)** 29](#_Toc87504657)

[**5.1. Como vai funcionar/ se monta a classe Servidor** 29](#_Toc87504658)

[**5.2.AÇÃO 1 DE INICIALIZAÇÃO** 30](#_Toc87504659)

[**5.2.1.PROMPT DE COMANDO SERVIDOR:** 31](#_Toc87504660)

[**5.2.2.PROMPT DE COMANDO CLIENTE:** 31](#_Toc87504661)

[**5.3.AÇÃO 2 DE INICIALIZAÇÃO** 33](#_Toc87504662)

[**5.4.AÇÃO 3 DE INICIALIZAÇÃO** 33](#_Toc87504663)

[**5.5. AÇÕES DE TAREFAS SIMULTÂNEAS** 33](#_Toc87504664)

[**5.5.1.NA AÇÃO 1 DAS TAREFAS SIMULTÂNEAS, TEMOS QUE** 34](#_Toc87504665)

[**5.5.2.NA AÇÃO 2 DE TAREFAS SIMULTÂNEAS, TEMOS QUE:** 34](#_Toc87504666)

[**5.5.3 NA AÇÃO 3 DE TAREFAS SIMULTÂNEAS, TEMOS QUE:** 35](#_Toc87504667)

[**AULA 6 – Fazedor de Continhas: Implementação das classes específicas do Servidor (parte 2)** 35](#_Toc87504668)

[**6.1.Arquivos usados:** 36](#_Toc87504669)

[**6.1.1.Arquivo: Servidor.java** 36](#_Toc87504670)

[**6.1.2.Arquivo: AceitadoraDeConexao.java** 37](#_Toc87504671)

[**6.1.3.Arquivo: SupervisoraDeConexao.java** 38](#_Toc87504672)

**Aulas sobre a Classe Servidor de Continhas**

1. **Arquivo: Aula Servidor de Continhas\_uma visão funcional.mp4**

**AULA 1 – Aula Servidor de Continhas: uma visão funcional**

Para usar o Sockets e Threads, vai ensinar um Sistema de Arquitetura Cliente/Servidor. É um fazedor de Continhas.

* 1. **Como funciona/vai ser montado o sistema de Arquitetura Cliente/Servidor**

**Basicamente ele funciona assim:**

1) Você ***roda o programa servidor e ele fica aguardando conexões de clientes***; caso seja digitado no servidor o comando “desativar”, o programa servidor enviará para todos os clientes conectados um aviso de que o servidor está sendo desligado e então o programa servidor termina sua execução; os programas clientes, ao receberem este aviso, informam ao usuário que o servidor está inoperante e pede para tentarem novamente mais tarde.

2) Você ***roda quantas vezes quiser o programa cliente e, toda vez que rodar o cliente, ele vai se conectar com o servidor; lá*** no servidor, existirá para ele uma variável double valendo zero; ele poderá então enviar solicitações para somar um valor a esta variável double , subtrair um valor dessa variável double, multiplicar essa variável double por um valor, dividir essa variável double por um valor, e todas essas operações farão ser alterado no servidor o valor daquela variável double que foi associada àquele programa cliente; o programa cliente poderá solicitar ainda no programa servidor o valor atual da variável double em questão e o programa servidor o enviará para o programa cliente que a receberá e exibirá na tela; finalmente o programa cliente pode pedir pata sair (desconectar) e o programa servidor toa as providências para a desconexão daquele programa cliente.

Quero que **examinem o programa com cuidado**, atenção, explorem, investiguem e tentem compreender o funcionamento dele. Em breve disponibilizarei para vocês um vídeo explicando o código dele.

***O servidor deve ser ligado antes;*** quando os clientes forem se conectar, o servidor vai ficar rodando. O servidor aguardar a conexão dos clientes; os clientes interagem com o servidor fazendo solicitações. O servidor atende e responde pro cliente. Quando o cliente estiver satisfeito, ele se desconecta. Os sockets entram para viabilizar a comunicação entre cliente e servidor. Eles vão estar rodando em 2 máquinas iguais ou diferentes e vão se comunicar pela rede; por isso que se usa sockets.

***As threads entram com o seguinte propósito:*** um servidor quando roda ele pode atender muitos clientes. Vários clientes podem ser ligados em diferentes máquinas rodando por aí e estarem conectados ao mesmo servidor, e obterem daquele servidor o serviço que eles desejam.

O servidor vai ser obrigado a atender simultaneamente uma série de clientes, pois nós não sabemos quando um cliente vai solicitar serviços ao servidor. O servidor tem que dar conta de todos os clientes, por isso que se usam as threads para que ele possa atender simultaneamente a todos os clientes. O sistema tem arquitetura cliente-servidor.

***Agora vou mostrar o sistema funcionando;*** vai mostrar o sistema cliente/servidor rodando. Pro sistema rodar direito, eu tenho que rodar primeiro o servidor e depois 1 ou mais clientes. Isso porque quando o cliente roda ele tenta se conectar no servidor; se o servidor não tiver rodando, provavelmente ele não vai conseguir se conectar e vai dar erro.

Para mostrar o erro, vai rodar somente o cliente.

**1.2. Lista de Arquivos usados nessa videoaula (Aula 1)**

**1.2.1.Arquivo: Cliente.java**

(09min32s)

Import java.net.\*;

Import java.io.\*;

Public class Cliente

{

public static final String HOST\_PADRAO = “ localhost”;

public static final int PORTA\_PADRAO = 3000;

public static void main (String [] args)

{

If (args.length>2)

{

System.err.println (“Uso esperado: java Cliente [HOST [PORTA]]\n”);

Return;

**Quando roda o arquivo cliente.java apenas**, ele tenta conectar e aparece uma mensagem no cmd: Indique a máquina e porta corretos! Ele dá a mensagem porque quando eu rodo o cliente, ele tenta se conectar no servidor: usa a máquina localhost e porta 3000 como padrões. Se eu quiser, eu posso dizer em qual máquina e porta devem ser conectadas. Se eu quisesse dizer a máquina e porta que deve ser conectada, eu usaria o comando: java Cliente [HOST [[PORTA]] onde host é a máquina e porta é a saída da porta; tanto host como porta, por estar em colchete são opcionais.

Se eu disser o host (nome da máquina ou ip da máquina) ele vai rodar pelo host que eu disse; se eu especificar a porta não será mais 3000 e sim o que tiver especificado. Como eu rodei sem especificar nada. Para funcionar mesmo, tem que rodar primeiro o Servidor.

**Ao rodar o Servidor, aparece a mensagem:** O servidor está ativo! Para desativá-lo, use o comando “desativar”. Na cmd, se escrever algo como umacoisaqualquer (que não seja o comando desativar), aparece comando inválido! O servidor está ativo! Para desativá-lo, use o comando “desativar”. Logo, o servidor foi programado para aceitar apenas o comando desativar; qualquer outra coisa escrita ele avisa estar errado e volta pra instrução de desativar. Se eu quisesse, poderia implementar um comando para saber quantos clientes estão conectados naquele momento; poderia extrair alguma informação do servidor.

Vamos rodar o cliente. Agora, com o servidor ativo, ele conseguiu rodar o cliente. Na prompt de comando do cliente aparece:

Sua opção (+, -, \*, /, = ,[T]erminar)?

***O cliente está conectado no servidor; se quiser, pode conectar mais clientes.*** O localhost é um cliente. Se rodar novamente cliente, serão 2 abas de prompt, o que significa ser 2 clientes conectados no servidor: são dois programas cliente rodando na mesma máquina. Quando o cliente se conecta no servidor; dentro do servidor vai ter a variável double para cada cliente que eu rodei. Cada cliente faz uma série de operações, mas inicialmente double é zero. Se eu quiser saber o valor da variável double tem no servidor, dou = e enter no cliente (aqui é cliente 2). Ao ter dado igual, o programa cliente 2 mandou pro servidor um pedido de resultado. O servidor, recebendo o pedido de resultado, mandou um resultado de volta ao programa cliente 2. O programa cliente 2 recebeu e printou na tela 0.0.

Se eu fizer = 10 no programa cliente 1, o resultado atual é 10.0 porque atualizou conforme a operação feita. São variáveis independentes, uma para cada cliente.

**1.2.1.1.Programa Cliente 1 (de cima):**

Sua opção (+, -, \*, /, = ,[T]erminar)? =

Resultado atual: 0.0

Sua opção (+, -, \*, /, = ,[T]erminar)? +

Valor? 10

Sua opção (+, -, \*, /, = ,[T]erminar)? =

Resultado atual: 10.0

Sua opção (+, -, \*, /, = ,[T]erminar)? =

Resultado atual: 10.0

Sua opção (+, -, \*, /, = ,[T]erminar)? /

Valor? 2

Sua opção (+, -, \*, /, = ,[T]erminar)? =

Resultado atual: 5.0

Sua opção (+, -, \*, /, = ,[T]erminar)? =

Resultado atual: 5.0

Sua opção (+, -, \*, /, = ,[T]erminar)?

O servidor vai ser desligado agora; volte mais tarde.

**1.2.1.2.Programa Cliente 2 (debaixo):**

Sua opção (+, -, \*, /, = ,[T]erminar)? =

Resultado atual: 0.0

Sua opção (+, -, \*, /, = ,[T]erminar)? =

Resultado atual: 0.0

Sua opção (+, -, \*, /, = ,[T]erminar)? –

Valor? 50

Sua opção (+, -, \*, /, = ,[T]erminar)? =

Resultado atual: -50.0

Sua opção (+, -, \*, /, = ,[T]erminar)? =

Resultado atual: -50.0

Sua opção (+, -, \*, /, = ,[T]erminar)? \*

Valor? -2

Sua opção (+, -, \*, /, = ,[T]erminar)? =

Resultado atual: 100.0

Sua opção (+, -, \*, /, = ,[T]erminar)? T

\*\*\*\*\*\*\*\*\*\*\*\*\*PROGRAMA CLIENTE 2 FECHOU SOZINHO\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**1.2.2.Arquivo: Servidor.java**

Import java.util.\*;

Public class Servidor

{

Public static String PORTA\_PADRAO = ‘‘3000’’;

Public static void main (String [] args)

{

If (args.length>1)

{

System.err.println (“ Uso esperado: java Servidor [PORTA]\n”)/

Return;

}

Agora quero mostrar que, **se eu termino um desses programas, uso a opção [t] para terminar**: o programa cliente 2 que pediu fecha, mas o restante (cliente 1 e servidor) continuam rodando. No servidor não aconteceu nada visível na tela. Mas tudo que o cliente 1 pediu para ser feito, foi feito pelo servidor, mas não foi mostrado na tela. São comunicações que o cliente e servidor fazem; as informações que o servidor manda pro cliente, o cliente printa.

Observe o que ocorre **quando pede para desativar o servidor no prompt de servidor**. O servidor vai ser parado (deu mensagem de o servidor foi desativado!), manda mensagem pro cliente, e o cliente avisa que o servidor está inoperante. Então, o servidor foi desligado e ao apertar qualquer tecla ele fecha o prompt; de igual modo, o programa cliente 1 está com servidor inoperante e quando clica em qualquer coisa ele fecha.

Vamos rodar novamente o servidor (24min15s) e o programa cliente 3.

**1.2.1.3.Na prompt de comando Programa Cliente 3:**

Sua opção (+, -, \*, /, = ,[T]erminar)? =

Resultado atual: 0.0

Sua opção (+, -, \*, /, = ,[T]erminar)? +7

Opcao invalida!

Sua opção (+, -, \*, /, = ,[T]erminar)? +

Valor? 7

Sua opção (+, -, \*, /, = ,[T]erminar)? =

Resultado atual: 7.0

Sua opção (+, -, \*, /, = ,[T]erminar)? \*

Valor? 2

Sua opção (+, -, \*, /, = ,[T]erminar)? =

Resultado atual: 14.0

Sua opção (+, -, \*, /, = ,[T]erminar)? /

Valor? 0

Valor invalido! // um número nunca pode ser divisível por zero; o cliente percebeu que dividir algo por zero e errado, deu valor invalido e nem mandou pro servidor.

Sua opção (+, -, \*, /, = ,[T]erminar)?

Sua opção (+, -, \*, /, = ,[T]erminar)?+

Valor? 3

Erro de comunicação com o servidor; tente novamente! Caso o erro persista, termine o programa e

volte a tentar mais tarde!

Sua opção (+, -, \*, /, = ,[T]erminar)? –

Valor? 2

Erro de comunicação com o servidor; tente novamente! Caso o erro persista, termine o programa e

volte a tentar mais tarde!

Sua opção (+, -, \*, /, = ,[T]erminar)? \*

Valor? 4

Erro de comunicação com o servidor; tente novamente! Caso o erro persista, termine o programa e

volte a tentar mais tarde!

Sua opção (+, -, \*, /, = ,[T]erminar)? /

Valor? 3

Erro de comunicação com o servidor; tente novamente! Caso o erro persista, termine o programa e

volte a tentar mais tarde!

Sua opção (+, -, \*, /, = ,[T]erminar)? t // como o erro persiste vou desligar o programa

Obrigado por usar este programa!

(Se pressionar qualquer tecla para continuar fecha esse programa).

O **servidor quando termina normalmente**, quando escreve desativar no servidor; ele manda mensagem aos clientes que avisa ao usuário do servidor desligado e pede para tentar em outro momento.

***E quando o servidor é interrompido de forma abrupta?*** Eu vou simular clicando no x, mas poderia ser uma queda de energia, queda de internet. No programa cliente 3 não aconteceu nada de aparente. Mas ao tentar inserir alguma operação vai dar erro.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**2) Fazedor de Continhas\_implementação das classes comuns ao Cliente e ao Servidor.mp4**

**AULA 2 – Fazedor de Continhas: Implementação das classes comuns ao Cliente e ao Servidor**

Nessa videoaula vai mostrar o código do sistema arquitetura cliente/servidor (fazedor de continhas). Vamos **examinar as classes que entram na composição daquele sistema**. Aquele sistema é constituído por dois programas: cliente e servidor. Existem algumas classes que fazem parte dos dois programas. As classes vão ser usadas nos meus programas para declarar e instanciar objetos que vão ser transmitir entre cliente-servidor (pode ser do cliente ao servidor e vice-versa).

São **6 classes que representam tudo o que pode ser transmitido entre um programa ao outro**; as classes estão relacionadas entre si através da herança. Elas formam uma hierarquia. O topo da hierarquia é a classe Comunicado, é de onde todas as outras classes herdam. Por enquanto, a classe Comunicado está com objeto oco (sem atributos nem métodos); não são todas as classes que tem objetos oco. A classe Comunicado é oca, enquanto algumas classes podem não estar ocas.

Para que os **objetos de certa sejam transmitidos através dos sockets** de um programa pro outro. Os objetos devem herdar da interface Serializable podem ser transmitidos através de Sockets, gravar objetos em arquivos, gravar objetos em banco de dados.

Quando quero fazer uma classe herdar de uma interface, uso a palavra implements no cabeçalho.

**2.1. Lista de Arquivos usados nessa videoaula (Aula 2)**

**2.1.1.Arquivo: Comunicação.java**

import java.io.\*;

public class Comunicado implements Serializable

{}

Agora vamos **verificar a classePedidoDeOperacao** que herda da classe Comunicado; e como Comunicado herda de Seriazible, o PedidoDeOperacao indiretamente também herda de serializable. O fazedor de continhas faz continhas; que são constituídas de operações, por exemplo, a continha de mais é uma operação de adição, enquanto que a de menos é uma operação de subtração. Então, quando o cliente quiser que o servidor faça uma operação, o cliente cria uma instância da classe PedidoDeOperacao ; para criar usa o construtor que passa qual a operação (+,-,\*, /) e o valor que deve ser o segundo operando dessa operação. O primeiro operando dessa operação é o valor que todos os clientes têm lá no servidor.

O **construtor de operação recebe operação e valor**, guarda a operação no atributo operação; pois os objetos da classe PedidoDeOperacao tem dois objetos: operação e valor. O construtor guarda a operação recebida no atributo operação e guarda o valor recebido no atributo valor. Após construir o PedidoDeOperacao, ele já pode ser mandado pelo cliente lá pro servidor. O servidor quando recebe o PedidodeOperação, realiza a operação.

**Quando ele recebe a operação e realiza**, ele precisa saber qual a operação que deve ser feita, e é por isso que temos o método getOperacao. O servidor quando receber uma operação, vai usar o getOperação para saber que operação vai usar; dentro desse método retorna this.operacao. De igual modo, o servidor precisa saber o valor que vai precisar retornar pelo método getValor.

**Quando vai fazer a operação de mais**: é um número (primeiro operando) mais outro número que é o segundo operando. O primeiro operando já está no servidor (é o double que começa com zero); a segunda operação é o cliente que manda. Na classe PedidoDeOperacao, temos o método toString.

Essas classes são, portanto, classes que podem ser usadas para declarar e instanciar objetos que vão ser **transmitidos do cliente pro servidor** e vice-versa.

**2.1.2.Arquivo: PedidoDeOperacao.java**

Public class PedidoDEOperacao extends Comunicado

{

Private char operação;

Private double valor;

Public PedidoDeOperacao (char operação, double valor)

{

this.operacao = operação;

this.valor = valor;

}

Public char getOperacao ()

{

return this.operacao;

}

public double getValor ()

{

Return this.valor;

}

Public String toString ()

{

Return (“” + this.operacao+this.valor);

}

A segunda classe é **PedidoDeResultado** é uma classe oca, que herda de Comunicado (herda indiretamente de Serializable. Pra essa classe ser usada, sempre que o cliente quiser saber qual é o resultado de uma operação , o valor que está guardado lá no servidor para ele; o cliente tem que mandar um objeto instanciado da classe PedidoDeResultado. O servidor quando recebe o objeto instanciado de PedidoDeResultado, ele entende que o cliente quer saber o valor que está guardado no servidor. O servidor responde mandando um objeto da classe Resultado.

Os **objetos da classe Resultado tem atributo valorResultante**; a classe Resultado herda de Comunicado e indiretamente herda da interface Serializable. Por ter o objeto valorResultante, é ele que será mandado ao cliente; é nesse objeto que se encontra o resultado do cliente no servidor. O construtor resultado recebe um valor e guarda o valor recebido no atributo this.valorResultante. Em resultado, temos o método getValorResultante que retorna o atributo valorResultante. Temos o método toString.

**2.1.3.Arquivo: PedidoDeResultado.java**

Public class PedidoDeResultado extends Comunicado

{}

**2.1.4.Arquivo: Resultado.java**

Public class Resultado extends Comunicado

{

Private double valorResultante;

Public Resultado (dobule ValorResultante)

{

this.valorResultante = valorResultante;

}

Public double getValorResultante ()

{

return this.valorResultante;

}

Public String toString ()

{

Return (“” + this.valorResultante);

}

}

Agora vai mostrar a classe PedidoParaSair.

**2.1.5.Arquivo: PedidoParaSair.java**

Public class PedidoParaSair extends Comunicado

{}

**2.1.6.Arquivo: ComunicadoDesDesligamento.java**

Public class ComunicadoDeDesligamento extends Comunicado

{}

A **classe PedidoParaSair herda de Comunicado** (diretamente) e Serializible (indiretamente); é uma classe oca. A classe ComunicadoDeDesligamento é uma classe oca que herda de Comunicado (diretamente) e Serializable (indiretamente). A maioria das classes são ocas; apenas as que não são ocas são PedidoDeResultado e PedidoDeOperação.

Quando o **usuário estiver usando o programa cliente e escolher a opção de terminar o programa**, o programa cliente vai mandar pro servidor uma instância da classe PedidoParaSair; para que o servidor saiba que o cliente está saindo e possa tomar todas as providências cabíveis dentro do servidor.

A **classe ComunicadoDeDesligamento**, terão os seus objetos usados quando no servidor o usuário escrever desativar. O servidor manda objetos da classe ComunicadoDeDesligamento para todos os clientes que tiverem conectados; para que todos os clientes saibam que o servidor está desligando e , portanto, os clientes não podem contar com os serviços do servidor. No servidor eu digito o comando desativar, o objeto será criado, instanciado e mandado pros clientes que ele será desligado.

**Os objetos das classes trafegam pela rede via sockets**. Exemplo, se o cliente mandar um PedidoDeOperação, essa mesma classe tem que existir no Servidor para que ele receba e o interprete. Servem para o cliente falar o que quer para o servidor e o servidor receber a solicitação do cliente e enviar de volta o que foi pedido. Poucas vezes o servidor quer alguma coisa, o servidor apenas atende os clientes.

Tem **uma situação que o servidor vai querer alguma coisa que é quando você digita o comando desativar;** o servidor vai querer informar para os clientes que ele será desligado e ficará inoperante. Logo, o servidor manda um objeto instanciado da classe ComunicadoDeDesligamento. A classe Resultado é o servidor que manda o objeto para o cliente. As demais classes são para gerar objetos que o cliente manda pro servidor; o servidor recebe e atende as requisições. As classes PedidoDeOperação, PedidoDeResultado e PedidoParaSair é o cliente que manda objetos das classes citadas para o servidor.

Agora vamos **falar sobre as outras duas classes: Teclado e Parceiro;** elas existem nos dois programas cliente e servidor. A classe Teclado nós já conhecemos bastante.; vai ser usada nos dois programas para permitir a digitação. No cliente, quando o usuário vai digitar a opção que ele deseja da operação: adição, subtração, divisão, multiplicação, resultado ou terminar; vai ser usada a classe Teclado. Quando o usuário escolhe as opções que representam uma operação (+,-,/,\*), vai ter que digitar o valor e depois o igual; nesses casos também se usam a classe Teclado. No servidor, quando digita o comando de desativar é usada a classe desativar.

Agora, temos a **classe Parceiro.**

**2.1.7.Arquivo: Teclado.java**

Import java.io.\*;

Public class Teclado

{

Private static BufferedReader teclado = new BufferedReader (new InputStreamReader(System.in);

Public static String getUmString ()

{

String ret = null;

try {}

catch () {}

}

**2.1.8.Arquivo: Parceiro.java**

Import java.io.\*;

Import java.net.\*;

Import java.util.\*;

Import java.util.concurrent.Semaphore;

Public class Parceiro

{

private Socket conexão;

private ObjectInputStream receptor;

private ObjectOutputStream transmissor;

private Comunicado proximoComunicado = null;

private Semaphore mutEx = new Semaphore (1,true);

public Parceiro (Socket conexão, ObjectInputStream receptor, ObjectOutputStream transmissor) throws Exception // se parâmetro for nulo

{

If (conexão ==null)

Throw new Exception (“ Conexão ausente”);

If (receptor == null)

Throw new Exception (“ Receptor ausente”);

If (transmissor == null)

Throw new Exception (“Transmissor ausente”);

this.conexao = conexão;

this.receptor = receptor;

this.transmissor = transmissor;

}

Public void receba (Comunicado x) throws Exception

{

try

{

this.transmissor.writeObject (x);

this.transmissor.flush ();

}

Catch (IOException erro)

{

throw new Exception (“Erro de transmissão”);

}

}

public Comunicado espie () throws Exception

{

try

{

This.mutEx.acquireUninterruptibly ();

If (this.proximoComunicado==null)

this.proximoComunicado = (Comunicado)this.receptor.readObject(); // if e this.proximocomunicado é uma operação só.

this.mutEx.release ();

return this.proximoComunicado;

}

catch (Exception erro)

{

throw new Exception (“Erro de recepção”);

}

Public Comunicado envie () throws Exception

{

try

{

if (this.proximoComunicado ==null)

this.proximoComunicado = (Comunicado)this.receptor.readObject(); if e this.proximocomunicado é uma operação só.

Comunicado ret = this.proximoComunicado;

this.proximoComunicado = null;

return ret

}

Catch (Exception erro)

throw new Exception (“Erro de recepção”);

}

public void adeus () throws Exception

{

try

{

this.transmissor.close();

this.receptor .close();

this.conexao .close();

}

catch (Exception erro)

{

throw new Exception ("Erro de desconexao");

}

}

}

A **classe Parceiro, também é usada no cliente e servidor**. Quando ela é usada no cliente, o parceiro representa o servidor; porque o servidor é o parceiro do cliente. Quando eu estiver implementando o servidor e estiver usando a classe Parceiro, lá no servidor a classe Parceiro vai servir para representar um dos vários clientes que vai estar se comunicando com o servidor. No cliente, sempre só tem um parceiro que é o servidor. No servidor, eu posso ter vários objetos da classe Parceiro porque vários clientes podem se conectar com o servidor. E os objetos da classe Parceiro no servidor representam o cliente.

**Como que a classe Parceiro foi feita?** Primeiramente ela tem como atributos o socket , o objectInputStream que vai estar associado ao mesmo socket e o objeto da classe ObjectOutputStream que vai estar associado também ao mesmo socket. Então, na classe parceiro vai ter conexão, receptor e transmissor que servem para receber e transmitir dados sempre através do socket conexao. Na classe Parceiro tem o objeto da classe Comunicado chamado proximoComunicado, ele serve para espiar qual é o comunicado que está vindo sem efetivamente consumir aquele comunicado que está chegando, sem tirar o comunicado do parceiro.

Depois, **na classe Parceiro, tem o objeto mutEx** que é uma abreviação de mútua exclusão; é da classe Semaphore , é instanciado como um novo semáforo que tem 1 recurso. Na hora que esse recurso for solicitado, qualquer outra solicitação vai gerar um bloqueio do solicitante; o solicitante que primeiro pedir vai ter o recurso zerado e o próximo que pedir vai ficar bloqueado enquanto o recurso for executado para depois ser liberado.

Depois dos objetos privativos**, temos o construtor de Parceiro,** que recebe conexão, receptor e transmissor como parâmetros. (Parei em 33min36s). Um parceiro é um programa que vai se comunicar com o nosso programa que está em uso. Logo que o construtor começa, os parâmetros são testados para ver se são nulos, se sim geram exceções de objetos ausentes. Se os parâmetros não derem exceções, o socket recebido vai ser guardado no atributo conexão; o objectinputstream será guardado em receptor e o object outputstream será guardado em transmissor.

Desta forma, **depois que o construtor executa, os objetos conexão, receptor e transmissor vão estar instanciados; o mutex vai estar instanciado** porque é feito isso na própria declaração com new. O objeto proximoComunicado não vai estar instanciado porque foi passado nulo na sua declaração e não foi instanciado no construtor.

Sobre o **método receba, ele tem como parâmetro um comunicado x**. O método espie que retorna um comunicado. O método envie que também retorna um comunicado. O método receba não serve para receber e sim para transmitir, porque quando eu estou no cliente e o parceiro é o servidor; e eu falo servidor receba. O servidor vai receber, pois o cliente vai transmitir. Logo, o método receba serve para o programa que usar transmitir e fazer o parceiro receber. Quando faço servidor.receba () eu estou transmitindo do cliente o objeto e fazendo o servidor receber.

O **método espie** serve para eu conseguir saber o que foi mandado pra mim sem consumir o que foi mandado para mim. O método envie serve para eu saber o que foi mandado; se eu uso esse método no programa cliente eu vou dizer servidor.envie () , ou seja, eu estou pedindo para o servidor enviar e o meu programa cliente vai receber o objeto. Logo, o método envie ele recebe os dados do outro programa. É uma questão de ponto de vista: sempre que alguém envia é claro que alguém recebe.

Sempre que alguém recebe é claro que é porque alguém enviou. Logo, toda recepção e todo o envio tem o ponto de vista contrário. Para facilitar o entendimento, preciso ver se estou dentro do cliente ou servidor e quem está chamando os métodos receba e envie para saber o que será feito dentro do programa ao qual me encontro.

O nome pode parecer estranho, mas tem que ver a lógica dentro do método para entender. **O método espie seria uma espécie de envie, mas sem consumir a informação.** Certas coisas que a gente faz gera um consumo, por exemplo, você está num programa lendo arquivos. Faz muitas leituras de informações, cada leitura te dá o que foi lido e é como se aquilo fosse consumido. Porque a próxima leitura vai te dar outra informação. É o que acontece no envie, ele vai fazer o programa receber alguma coisa, e na próxima vez que eu chamar o envie, o programa vai receber outra coisa. Se estou dentro do programa cliente e faço servidor.envie, o servidor vai enviar informações para que eu, cliente, receba. Então, o envie é um transmissor de dados. Desta forma, a cada vez que eu chamar o envie, o meu programa vai receber dados sempre diferentes da rodada anterior.

O **método espie ele me diz o que tem lá para eu receber, mas não consome.** Se eu espio para ver o que tem eu vejo uma coisa, se espiar de novo eu vejo a mesma coisa. Se espiar pela 3ª vez será visto a mesma coisa. Somente quando eu chamar envie que de fato o meu programa recebeu; se chamar o envie novamente recebo outra coisa. É como se fosse uma espiada no futuro, quando eu for precisar chamar o envie vou receber x porque uso o espie. O espie não deixa que eu tire o que vou receber, isto é, não faz o envie funcionar.

**Como o método espie foi feito?** Usa o atributo proximoComunicado que começa como nulo. Quando eu chamar o espie, ele verifica se o comunicado é nulo; ele vai fazer o proximocomunicado receber e deixar de ser nulo. O proximocomunicado recebe receptor.readObject. Logo, o objeto receptor vai ser lido, convertido em comunicado e guardado no objeto proximocomunicado que deixará de ser nulo. Depois, eu retorno o que está guardado em proximoComunicado. A próxima vez que chamar o espie, como não é nulo, não usa readobject porque não entra no if; eu retorno a mesma coisa que tinha retornado antes.

**Quando eu chamar o método envie; ele vai ser se o proximocomunicado for nulo**, ele recebe a leitura do objeto receptor e guarda em proximocomunicado. Se o proximocomunicado não for nulo, ele guarda a leitura do proximocomunicado no objeto ret. Deixa o proximocomunicado nulo e retorna ret. Então, ele retorna ou o que acabou de ser lido quando proximocomunicado for nulo, ou ele retorna o que já estava lido. Sempre que eu chamo envie é porque o meu programa vai de fato receber, e o proximocomunicado fica nulo.

**O método receba tem como parâmetro um comunicado**, serve para fazer o meu programa enviar e o meu parceiro receber. Então, quando o meu parceiro recebe, o meu programa envia. O meu programa chama o receba porque ele (meu programa) quer enviar e deseja que o parceiro receba. Como esse método funciona? Pega o parâmetro (transmissor) e faz writeObject (x) e depois um flush para ser realmente enviado o transmissor por aquele objeto.

**Dentro do método espie estou usando o semáforo mutEx que significa mútua exclusão**. Porque na hora que eu for ler alguma coisa do proximoComunicado, eu estou trabalhando em ambiente de threads. Então, tem coisas acontecendo ao mesmo tempo. Eu não quero que uma thread teste que o proximocomunicado é nulo, não deu tempo dela ler e foi de repente interrompida. Ao mesmo tempo, outra thread entrou em execução e encontra o proximocomunicado nulo, lê. Depois a primeira thread volta para executar, e lê por cima. Logo, para evitar essa situação de conflito entre duas threads, antes dela testar se é nulo e ler, eu requisito um recurso do semáforo mutEx e depois de ter visto se é nulo e lido, eu libero um recurso com a chamada do método release no semáforo mutEx (mutEx.release ()).

Como **o semáforo mutEx foi instanciado com novo semáforo com 1 recurso.** Suponha que tenha várias threads espiando. Na hora que uma thread espiar e pedir o recurso, o recurso fica zero. Essa thread consegue checar se é nulo e mesmo que perca a vez de executar e uma segunda thread entrar em execução; se a segunda thread pedir recurso será bloqueada pela falta de recurso.

Depois, **a primeira thread volta a executar e lê o objeto receptor guardando em proximoComunicado**. Tanto o teste do if e a leitura do proximoComunicado quero que seja uma operação unitária, usando o mutEx como semáforo para evitar de 2 threads tentar fazer a mesma coisa e uma delas não ter tempo de fazer as coisas.

No **fim da classe Parceiro nós temos o método adeus que serve para desconectar com o parceiro.** Então, se o cliente chamar servidor.adeus (), o cliente está desconectando do servidor. Pois o adeus fecham os canais de transmissor, receptor e conexão; desta forma vai estar interrompendo a conexão com o parceiro. Isso é feito quando o programa está terminando.

**3) Arquivo: Fazedor de Continhas\_implementação das classes específicas do Cliente (parte 1).mp4**

**AULA 3 – Fazedor de Continhas: Implementação das classes específicas de CLIENTE (parte1)**

**3.1. Como funciona/vai ser montada a classe Cliente**

**CLIENTE**

**AÇÕES DE INICIAÇÃO QUE DEVEM SER REALIZADAS NO CLIENTE:**

**ESSAS AÇÕES SÃO REALIZADAS NO INÍCIO DA MAIN DO CLIENTE**

1) Checar os parâmetros vindos para main em args para permitir que o usuário indique a máquina onde roda o Servidor e/ou a porta adotada pelo Servidor, ao qual o Cliente deve se conectar, sinalizando eventuais erros, ou, caso nenhum parâmetro venha para a main em args, assumir que o servidor roda em uma máquina padrão e que ele adotou uma porta padrão.

2) Realizar a conexão com o Servidor, instanciando um transmissor (ObjectOutputStream) e um

Receptor (ObjectInputStream) para, com tudo isso, (conexão, transmissor e receptor) instanciar um Parceiro servidor, que será usado ao longo de todo o programa para se comunicar com o Servidor;

3) “Startar” uma TratadoraDeComunicadoDeDesligamento, naturalmente compartilhando com ela o Parceiro servidor, de onde ela receberá, EVENTUALMENTE, um ComunicadoDeDesligamento.

O objetivo dessa videoaula é se ***aprofundar na implementação do cliente-servidor*** que constituem o fazedor de continhas. Tem que compreender a fundo como foi implementado tanto o cliente como o servidor do sistema cliente-servidor. Mesmo sendo um sistema simples, ele é um cliente do sistema

Cliente-servidor que costuma manter os sistemas cliente-servidor. Portanto, se compreender bem como foi implementado, isso vai nos agregar conhecimentos que permitirão no futuro, implementar outros sistemas cliente-servidor com funcionalidades bem mais sofisticadas, mas que não fogem da estrutura que nós vemos aqui.

O ***sistema cliente-servidor tem uma estrutura padrão e agora vamos conhecer;*** pode servir de base aos demais sistemas cliente-servidor. Nessa videoaula vai focar no cliente e na aula seguinte a do servidor.

As ***funcionalidades principais que devem ser realizadas pelo cliente no cliente servidor.*** Temos ações realizadas na main no início do cliente, acontecem assim que põe para rodar o cliente. Cada item enumerado é uma ação.

**Na ação 1,** máquina e porta podem estar indicadas no vetor args pelo usuário no programa Cliente. Não pode ter mais de 2 informações no args. Me comunico com a máquina na porta que for indicada.

**Na ação 2**, o cliente se conecta com o servidor (declarar e instanciar um objeto da classe socket), o construtor de socket exige a máquina e porta que deseja conectar (clientes proativos, tomam a iniciativa de se conectar ao servidor que é reativo). Ou seja, usa máquina e porta da ação 1.

***Depois no item 2, instanciam e declaram o transmissor (passa os dados da conexão) , o receptor (passa os dados da conexão).*** Depois, instancia e declara o objeto da classe Parceiro. Eu estou no programa cliente e a conexão que eu tenho é com o servidor: o transmissor é para enviar ao servidor, e o receptor é para receber do servidor. O parceiro que será criado vai ser um parceiro que vai permitir comunicação com o servidor. Classe parceiro tem os 3 métodos: envie, receba e espie; eles se comunicam com o servidor.

***Na ação 3, precisa ser declarada, instanciada e startada uma thread para tratar o comunicado de desligamento*** (classe TratadoraDeComunicadoDeDesligamento, ela herda da classe Thread), dessa classe eu posso chamar o método start para startar. Quando o método start da thread é chamado ela entra em execução e começa a executar o método run e faz todo o trabalho que espera que seja feito da classe TratadoraDeComunicadoDeDesligamento. É importante que a main vai precisar compartilhar com a thread da classe TratadoraDeComunicadoDeDesligamento o parceiro servidor que foi obtido na ação 2. ( A ação 2 termina com a declaração e instanciação do parceiro servidor).

Ao **longo do programa cliente inteiro só vai haver comunicação com 1 servidor**; então, se na ação 2 eu criei o programa parceiro que me permitiu comunicar com o servidor; é obvio que essa TratadoraDeComunicadoDeDesligamento ao precisar se comunicar com o servidor ela vai se comunicar com o mesmo servidor e isso faz com que a main compartilhe com a thread o parceiro servidor.

**Como que se efetiva o compartilhamento?** A main declarou o objeto chamado parceiro servidor, instanciou usando a conexão, transmissor e recepctor; a main tem esse parceiro servidor declarado e instanciado. Na hora que a main for pra instanciar o objeto da classe TratadoraDeComunicadoDeDesligamento, o construtor dessa última classe aceita receber um parâmetro que vai ser do tipo parceiro. E é ali na main quando for declarar e instanciar o objeto da classe TratadoraDeComunicadoDeDesligamento vai passar pro construtor o parceiro servidor que ela tem; porque na ação 2 ela declarou e instanciou o parceiro servidor. Logo, a classe TratadoraDeComunicadoDeDesligamento já vai ter acesso a esse parceiro servidor passado no construtor.

**É através desse parceiro servidor que ela vai receber eventualmente (nem sempre), um ComunicadoDeDesligamento.** Pode acontecer as vezes de um cliente se conectar com o servidor, fazer todos os usos de serviço que ele desejaria fazer do servidor, desconectar do servidor e não receber nunca o ComunicadoDeDesligamento. O ComunicadoDeDesligamento acontece quando o usuário administrador do servidor, lá no servidor, escreve o comando desativar. Na hora que, o usuário administrador dentro do programa servidor escreve desativar, ele está decidindo desligar o servidor. Nessa hora, o servidor vai mandar para todos os clientes conectados um comunicadodedesligamento.

Pode **acontecer de um certo cliente, se conectar no servidor, fazer o que precisava para se comunicar com o servidor; sair do programa cliente** porque o usuário do cliente resolveu terminar o programa. E nesse tempo em que esteve com o servidor não acontecer de chegar nenhum comunicadodedesligamento, pois nesse período de tempo o administrador não pediu o comando desativar no servidor.

**Pode acontecer de algum cliente receber o ComunicadoDeDesligamento**, e justamente por isso que existe a thread (que é a classe) TratadoraDeComunicadoDeDesligamento ; ela fica de olho naquilo que o servidor manda para ver se foi mandado um ComunicadoDeDesligamento. Se foi mandado esse comunicado, então, essa thread vai ter que fazer todas as ações necessárias para o cliente terminar a execução do programa cliente; uma vez que o cliente só funciona comunicando-se com o servidor. Se o servidor foi desativado, os clientes não tem como continuar executando.

Logo, a **classe TratadoraDeComunicadoDeDesligamento vai permitir que o cliente termine a sua execução**, mesmo que o usuário cliente não opte por terminar. Pois o programa cliente termina quando o usuário cliente pede a opção t para terminar. (26min26s).7

**Depois que a main fez essas ações de iniciação, ela não termina;** a main entra num loop e executa o loop junto com a thread T|ratadoraDeComunicadoDeDesligamento porque a main startou, então quando a main estiver na parte final dela; após as ações de iniciação, ela vai executar um loop junto com o objeto declarado e instanciado da classe TratadoraDeComunicadoDeDesligamento. (parei em 27min26s).

**TIPOS DE TAREFAS SIMULTÂNEAS QUE DEVEM SER REALIZADAS NO CLIENTE APÓS A REALIZAÇÃO DAS AÇÕES DE INICIAÇÃO LISTADAS ACIMA:**

**ISSO TUDO É REALIZADO NO FINAL DA MAIN DO CLIENTE, QUE FICA EXECUTANDO JUNTO COM A THREAD, APÓS EXECUTAR AS TAREFAS DE INICIALIZAÇÃO. (VÁLIDO PARA AÇÃO 1 ADIANTE).**

1. CONTINUAMENTE interagir com o usuário, mostrando as opções que eles têm, solicitando um valor que funcionará como segundo operando, caso ele tenha optado por fazer um PedidoDeOperação, ou mostrando Resultado, caso ele tenha optado por fazer um PedidoDeResultado, e sempre sinalizando erros por mal uso;

**ISSO TUDO É REALIZADO PELA THREAD TratadoraDeComunicadoDeDesligamento (VÁLIDO PARA A AÇÃO 2 ADIANTE).**

1. CONTINUAMENTE aguardar um EVENTUAL ComunicadoDeDesligamento vindo do Servidor, para então tomar as medidas necessárias do lado do Cliente.

A **ação 1 das tarefas simultâneas será realizada** (dentro da main) no fim da main do cliente. A outra (ação 2 das tarefas simultâneas) será executada pela thread tratadoraDeComunicadoDeDesligamento .

**O que será executada na main no fim da main depois das inicializações?** A main vai continuamente interagir com o usuário, significa ter um loop que vai acontecer sempre interação com o usuário. As interações são: opções que o usuário tem (operações, solicitação de resultado, terminar), solicitar valor depois de ter escolhido operações no pedidoDeOperação (precisa de 2 operandos). As opções são operações aritméticas, o primeiro valor está no servidor que é o double; mas para fazer operação precisa de 2 valores por isso que se pede o segundo operando. Logo, a solicitação de valor é o segundo operando. As operações produzem um resultado que vai ser guardadas dentro daquele número no servidor, com isso aquele primeiro operando é atualizado quando se pede o resultado. Depois de fazer 1 ou mais operações ele vai querer o resultado, por isso solicita o PedidoDeResultado e o servidor manda de volta o resultado para o cliente que printa na tela.

As **classes PedidoDeOperação, Resultado, PedidoDeResultado são classes já vistas quando estudamos a classe Comunicado e as classes que herdam dela**. Todas as classes citadas herdam da classe Comunicado. A classe PedidoDeOperação tem 2 atributos: operação que o usuário escolheu (char +, -, \*, /) e o número int chamado de valor que será solicitado na digitação. Esse objeto valor será mandado pro servidor, recebe o objeto, pega o número e a char operação; faz a operação tendo como primeiro operando o valor guardado em referência ao cliente, e o segundo operando o valor recebido da classe PedidoDeOperaçao. Logo, ele atualiza o número daquele cliente.

**Quando o cliente quiser saber o resultado,** ele declara, instancia e manda para o servidor um objeto da classe PedidoDeResultado (classe oca, sem atributos e sem métodos); o servidor identifica a classe. Então, o servidor declara, instancia e manda pro cliente o objeto da classe Resultado (tem o valorResultante) que é o resultado (tem atributos, métodos, construtor aceita valor de resultado). Chegou no cliente, ele recebe e mostra na tela.

**A main no final da sua execução, após as inicializações, ela faz a ação 1 de interação com cliente de forma contínua.** Uma das opções mostradas pro usuário é a opção t de terminar, o programa termina; e senão pedir operações aritméticas, resultado e nem t; o programa sinaliza mal uso por não ter essa opção desejada.

**Na ação 2 (depois das inicializações da main), precisamos examinar a parte 2 das tarefas simultâneas que são realizadas no cliente.** O que está sendo explicado no item 2 é realizado na thread TratadoraDeComunicadoDeDesligamento; que foi instanciada na ação 3 do início do main. Agora, a thread já startada roda junto com a main, e a thread fazem loop a verificação de eventual ComunicadoDeDesligamento (se virá e quando virá) ele vem do servidor. E essa nossa thread precisa estar continuamente aguardando a chegada dele; quando chegar, tomas as ações necessárias do lado do cliente: avisa o cliente do servidor desligado , programa será desligado, estimular o usuário a voltar mais tarde.

**As medidas necessárias constituem de:** avisar o usuário de que o programa será terminado porque o servidor foi desligado, o programa cliente vai terminar também. Então o programa cliente termina quando o servidor termina.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**4) Arquivo: Fazedor de Continhas\_implementação das classes específicas do Cliente (parte 2).mp4**

**AULA 4 – Fazedor de Continhas: implementação das classes específicas do CLIENTE (parte 2)**

**4. Lista de Arquivos usados na videoaula (aula 4)**

**4.1.Arquivo: Cliente.java**

Import java.net.\*;

Import java.io.\*;

Public class Cliente

{

public static final String HOST\_PADRAO = "localhost";

public static final int PORTA\_PADRAO = 3000;

public static void main (String[] args)

{

if (args.length>2)

{

System.err.println ("Uso esperado: java Cliente [HOST [PORTA]]\n");

return;

}

Socket conexao=null;

try

{

String host = Cliente.HOST\_PADRAO;

int porta= Cliente.PORTA\_PADRAO;

if (args.length>0)

host = args[0];

if (args.length==2)

porta = Integer.parseInt(args[1]);

conexao = new Socket (host, porta);

}

catch (Exception erro)

{

System.err.println ("Indique o servidor e a porta corretos!\n");

return;

}

ObjectOutputStream transmissor=null;

try

{

transmissor =

new ObjectOutputStream(

conexao.getOutputStream());

}

catch (Exception erro)

{

System.err.println ("Indique o servidor e a porta corretos!\n");

return;

}

ObjectInputStream receptor=null;

try

{

receptor =

new ObjectInputStream(

conexao.getInputStream());

}

catch (Exception erro)

{

System.err.println ("Indique o servidor e a porta corretos!\n");

return;

}

Parceiro servidor=null;

try

{

servidor =

new Parceiro (conexao, receptor, transmissor);

}

catch (Exception erro)

{

System.err.println ("Indique o servidor e a porta corretos!\n");

return;

}

TratadoraDeComunicadoDeDesligamento tratadoraDeComunicadoDeDesligamento = null;

try

{

tratadoraDeComunicadoDeDesligamento = new TratadoraDeComunicadoDeDesligamento (servidor);

}

catch (Exception erro)

{} // sei que servidor foi instanciado

tratadoraDeComunicadoDeDesligamento.start();

char opcao=' ';

do

{

System.out.print ("Sua opcao (+, -, \*, /, =, [T]erminar)? ");

try

{

opcao = Character.toUpperCase(Teclado.getUmChar());

}

catch (Exception erro)

{

System.err.println ("Opcao invalida!\n");

continue;

}

if ("+-\*/=T".indexOf(opcao)==-1)

{

System.err.println ("Opcao invalida!\n");

continue;

}

try

{

double valor=0;

if ("+-\*/".indexOf(opcao)!=-1)

{

System.out.print ("Valor? ");

try

{

valor = Teclado.getUmDouble();

System.out.println();

if (opcao=='/' && valor==0)

{

System.err.println ("Valor invalido!\n");

continue;

}

}

catch (Exception erro)

{

System.err.println ("Valor invalido!\n");

continue;

}

servidor.receba (new PedidoDeOperacao (opcao,valor));

}

else if (opcao=='=')

{

servidor.receba (new PedidoDeResultado ());

Comunicado comunicado = null;

do

{

comunicado = (Comunicado)servidor.espie ();

}

while (!(comunicado instanceof Resultado));

Resultado resultado = (Resultado)servidor.envie ();

System.out.println ("Resultado atual: "+resultado.getValorResultante()+"\n");

}

}

catch (Exception erro)

{

System.err.println ("Erro de comunicacao com o servidor;");

System.err.println ("Tente novamente!");

System.err.println ("Caso o erro persista, termine o programa");

System.err.println ("e volte a tentar mais tarde!\n");

}

}

while (opcao != 'T');

try

{

servidor.receba (new PedidoParaSair ());

}

catch (Exception erro)

{}

System.out.println ("Obrigado por usar este programa!");

System.exit(0);

}

}

**Agora vai falar sobre a execução de um programa.** Nós que somos programadores, estamos acostumados a rodar programas que estejam dentro do ambiente de desenvolvimento. Aqui dentro, clica nos botões de compilar e executar e fazem o que é pedido. Essa maneira de executar programa só é possível para o desenvolvedor e não para o usuário. Porque não tem acesso ao ambiente de desenvolvimento. Os usuários rodam o programa da seguinte maneira: tenho o prompt de comando D:\> significa que o prompt de comando está pronto para aceitar um comando. O que está escrito na cmd é o prompt.

**Esse prompt tem um significado, o D:** informa que o disco que está sendo usado é o D; os dois pontos é a pasta raiz que tem o mais alto nível hierárquico. Eu escrevo o nome do disco : e dou enter (para mudar a pasta raiz); então mudei de D: para C: . No prompt apareceu C:\>

Se eu quiser ir para outra pasta, escrevo cd (change directory) e escrevo para qual pasta quero ir:

“Fazedor de Continhas Cliente-Servidor” e dou enter.

Apareceu : C:\ Fazedor de Continhas Cliente-Servidor>

**O prompt de comando está me dizendo que o disco corrente ao disco C** e a pasta corrente é a pasta continhas cliente-servidor. (04min02s). Agora digamos que eu quero entrar na pasta servidor, ou usar:

C:\Fazedor de Continhas Cliente-Servidor > CD Servidor (enter)

C:\ Fazedor de Continhas Cliente-Servidor\ Servidor >

**Agora estou na pasta servidor que fica dentro da pasta fazedor de continhas cliente-servidor.** Qualquer comando que eu fizer na pasta servidor terá relação com a pasta fazedor de continhas cliente-servidor. Depois de servidor, posso escrever:

C:\ Fazedor de Continhas Cliente-Servidor\ Servidor > JAVA Servidor (enter)

O servidor esta ativo! Para desativa-lo, use o comando “desativar”

**Rodei o servidor. O interessante é que dá para usar o prompt de comando para fornecer informações para o programa Java Servidor.** Eu mandei rodar o servidor.

Vou fazer:

Desativar

O servidor foi desativado!

Eu saí do programa

C:\ Fazedor de Continhas Cliente-Servidor\ Servidor > JAVA Servidor 7000 (enter)

**Vai entrar no programa servidor do mesmo jeito, mas escrevo 7000 para entrar no vetor args do servidor.** O servidor assim como o cliente tem uma main, e essa main tem args. Quando escrever o 7000, ele vai aprar dentro do vetor args e poder ser usado.

**Depois da linha Java Servidor 7000, ele deu enter e apareceu a seguinte mensagem**: “O servidor está ativo! Para desativá-lo, use o comando “desativar”. O servidor está rodando e usando a porta 7000.

**Vamos rodar o cliente, foi executar o cliente.java**

No prompt:

D:\>

D:\>C:

C:\ Fazedor de Continhas Cliente-Servidor> cd Cliente

C:\Fazedor de Continhas Cliente-Servidor\Cliente> java Cliente (enter)

Quando o servidor está rodando e eu rodo o cliente, tudo funciona muito bem; se fosse ao contrário daria problema de escolher máquina e porta certa. Agora não deu certo.

Apareceu:

Indique o servidor e a porta corretos.

C:\Fazedor de Continhas Cliente-Servidor\Clientes

**Deu erro porque no servidor eu indiquei a porta 7000; faltou a porta no cliente.** O cliente usou a porta 3000 (padrão) e não deu certo.

Eu tenho que dizer na hora de rodar o cliente, dizer a máquina e a porta que quero usar

C:\Fazedor de Continhas Cliente-Servidor\Cliente> Cliente localhost 7000

Agora vai dar certo, apareceu:

Sua opção (+,-, \*, \, =, [T]erminar)? t

Obrigado por usar esse programa!

**Agora ele achou o cliente, por ter achado a porta e máquina certa.** Desativou o servidor (comando desativar) e no prompt do cliente colocou t para terminar. Quando no servidor não especificava a porta, não precisava especificar no cliente e era sempre localhost 3000.

Então:

**4.2.Prompt servidor:**

C:\ Fazedor de Continhas Cliente -Servidor\ Servidor>JAVA Servidor // usa o padrão porta 3000

O servidor esta ativo! Para desativa-lo, use o comando “desativar”

>desativar

O servidor foi desativado!

**4.3.Prompt Cliente:**

C:\ Fazedor de Continhas Cliente-Servidor\Cliente>JAVA Cliente // rodo o cliente sem especificar porta, ele assume que vai se comunicar na porta 3000 e máquina local.

Sua opção (+,-, \*, \, =, [T]erminar)? T

Obrigado por ter usado o programa!

Os dois assumiram algo compatível.

Se eu terminar servidor e cliente, rodo o servidor novamente ; e depois rodo cliente.

**4.4.Prompt Servidor:**

C:\ Fazedor de Continhas Cliente -Servidor\ Servidor>JAVA Servidor

O servidor esta ativo! Para desativa-lo, use o comando “ desativar”

>desativar

O servidor foi desativado!

C:\ Fazedor de Continhas Cliente -Servidor\ Servidor>JAVA Servidor

O servidor esta ativo! Para desativa-lo, use o comando “ desativar”

>desativar

O servidor foi desativado!

C:\ Fazedor de Continhas Cliente -Servidor\ Servidor>cls

O comando cls limpa a tela.

**Vou rodar o servidor na porta 7000.**

**4.5.No prompt servidor:**

C:\ Fazedor de Continhas Cliente -Servidor\ Servidor> Java Servidor 7000

O servidor esta ativo! Para desativa-lo, use o comando “ desativar”

**Se eu mandar rodar o cliente sem falar a porta, vai dar erro porque o servidor está na porta 7000 enquanto que o servidor está na porta 7000.** Logo:

**4.6.Prompt Cliente:**

C:\ Fazedor de Continhas Cliente-Servidor\Cliente> Java Cliente

Indique o servidor e a porta corretos!

C:\ Fazedor de Continhas Cliente-Servidor\Cliente> Java Cliente localhost 7000

Ou seja, eu coloco informações para que o cliente funcione.

Localhost vai pro vetor args na posição 0, e 7000 vai pro mesmo vetor na posição 1.

**Na prompt servidor**, quando coloquei Java Servidor 7000, o 7000 foi pro vetor args na posição 0.

**Na prompt cliente**, se eu colocar mais coisa do que o pedido depois de localhost 7000:

**4.7.Prompt Cliente**

C:\ Fazedor de Continhas Cliente-Servidor\Cliente> Java Cliente localhost 7000 bobagens (parei em 14min30s).

Além de colocar a porta, colocou uma bobagem na frente. Aconteceu o que se observa a seguir.

**4.8.No prompt Cliente:**

Uso esperado: java Cliente [HOST [PORTA]]

C:\ Fazedor de Continhas Cliente-Servidor \Cliente

Ele espera como resposta a máquina e a porta; portanto deu erro. Se colocasse como abaixo:

C:\ Fazedor de Continhas Cliente-Servidor \Cliente > JAVA Cliente localhost 7000

Sua opção (+,-, \*, /, = , [T]erminar)?

Então, **combinou cliente e servidor pelo localhost e porta que tem as mesmas configurações em ambos.** Tudo é programado na main dos dois. No vetor args somente aceita duas coisas. No programa Cliente, texto se args.length for maior que 2 vai dar o erro de uso inesperado; e depois dá o return para sair do programa.

**Senão for maior do que 2 args.length**, ele aceita entrar no programa Cliente e se conectar com o servidor. Entra no try, coloca a constante de host\_padrão o localhost; e a constrante porta\_padrao coloca a porta que foi passada. O uso do final em ambos faz ser constante.

**Se args.length for maior do que zero (tal como localhost e 7000),** pois args.length é 2 e 2>0. Então, entra no primeiro if para colocar a variável host que armazena cliente.host\_padrão na posição zero do vetor args. Se args.lengths for realmente 2, pega a segunda coisa do vetor args e transforma em int e guarda na variável porta que recebe cliente.porta\_padrao; cuja porta ficará na posição 1 do vetor args (depois de ser convertido de string para int na variável porta pelo método parseInt de Integer). Depois, com as duas posições preenchidas, faz o objeto conexão que será criado, instanciado e declarado como new Socket que recebe como parâmetro host e porta.

**Agora, imaginem que eu não coloque nada na hora de rodar o servidor.** Desativo os programas servidor e cliente. E no servidor não passo parâmetro e farei o mesmo para o cliente. Logo:

**4.9.Prompt Servidor**

>desativar

O servidor foi desativado!

C:\ Fazedor de Continhas Cliente-Servidor\ Servidor> JAVA Servidor (enter)

O servidor esta ativo. Para desativa-lo use o comando “desativar”

**4.10.Prompt Cliente**

C:\Fazedor de Continhas Cliente-Servidor\ Cliente> JAVA Cliente localhost 7000

Sua opção (+,-,\*,/,=, [T]erminar)?

O servidor vai ser desligado agora;

Volte mais tarde!

C:\

C:\Fazedor de Continhas Cliente-Servidor\ Cliente> JAVA Cliente

Sua opção (+,-,\*,/,=, [T]erminar)?

**Não passar parâmetro de local de máquina e nem porta nos dois programas fez com que os dois rodassem porque ambos adotam o padrão da main.** Sei que ele vai usar a porta 3000 porque não tem nada escrito depois de Java Cliente. Ao analisar o programa Cliente.Java, dentro da main. Tem dentro de public static void main (String [] args) um primeiro if para testar se o comprimento de args é maior que 2, o que é mentira; pois args.length é zero pela falta de passar parâmetro do usuário cliente.

**Como não entrou no if, ele segue o programa e testa o try;** onde a variável string host recebe cliente.host\_padrao (cujo atributo na main é localhost); depois vai colocar na variável int porta que recebe cliente.porta\_padrao (cujo atributo na main é 3000).

**Foi colocado nas variáveis os valores padrão da main por enquanto.** Depois, dentro do try, o programa verificar se o comprimento de args é maior do que zero; porém zero não pode ser maior que ele mesmo então não entra no if. Por isso, não muda o host. Como args.length =0, no segundo if verifica se o comprimento de args é maior do que dois, como zero é menor do que 2 então, não entra no if e não muda a porta.

**Por isso que o arg mantém os valores originais de localhost e porta que serão usados no objeto conexão.** Se eu colocar algo na frente do número da porta; ele passa o valor de localhost para host, o valor porta para a variável porta. Mas no primeiro if muda o localhost e no segundo if muda a porta. Porém, ficou sobrando o bobagens que dá erro.

**Portanto, inicialmente o programa sempre passa padrão, mas se o comprimento for igual a 2,** ele transcreve o que foi passado no prompt para os comandos da main, de modo que se alterem a máquina e a porta. Depois, conecta pelos sockets com os últimos dados passados. (24min17s). Então, o meu programa pode ser ativado pelo localhost e porta padrão ou pelos dados que eu passei para que a main assuma os valores.

**E isso que ele acabou de mostrar é o item 1 da inicialização** (parei em 25min46s).

**Agora vai mostrar no código o item 2 da inicialização**. A conexão com o servidor ocorre depois do vetor args, onde tem conexão = new Socket (host, porta); seja com o padrão ou o que foi indicado no args, faz a conexão. Após a conexão, ocorre a declaração do transmissor e a sua instanciação usando a conexão com o servidor. Por isso o transmissor vai transmitir com o servidor. Após o transmissor, fazemos a declaração e instanciação do receptor que uso a conexão com o servidor; por isso o receptor vai receber coisas do servidor. Depois, declaramos e instanciamos o parceiro servidor; ele é instanciado com um novo parceiro passando para o construtor a conexão, transmissor e receptor.

**Para o item 3 de inicialização,** vai mostrar o startar da classe TratadoraDeComunicadoDeDesligamento. O item 3 ocorre após o new Parceiro. Declaramos a TratadoraDeComunicadoDeDesligamento como um objeto nulo. Em seguida faz o try com a instanciação do objeto tratadoraDeComunicadoDeDesligamento com new tratadoradecomunicadodedesligamento e passa como parâmetro para essa classe o servidor. Após chama o método start para o mesmo objeto tratadoradecomunicadodedesligamento.

***Agora vamos entrar nas tarefas simultâneas: são 2 tarefas que são executadas.***

***Para a tarefa 1 simultânea, temos que no arquivo: Cliente.java,*** depois de ter feito o start da thread, tem a declaração do objeto opção que recebe aspas simples do tipo char. Após, tem o loop com do onde mostra a opção e diz as possibilidades: +,-,\*, /, =, [T]erminar ? Ele pergunta a opção do usuário. Em seguida, no try usa a classe Teclado com o método getUmChar que são os parâmetros para Charactere.toUpperCase para o objeto opção. Senão for a opção desejada, aparece no catch o erro de opção inválida. O toUpperCase é para ver se o usuário teclou alguma letra vai ficar maiúsculo, como o t virar T. O método getUmChar dá exceção quando não digitou um char. Depois do catch (dentro dele) vem o comando continue, no qual exige que não se execute nada que vem pela frente e volta no começo do loop com repetição nova, isso quando o usuário for no catch.

***Se a pessoa não errou, pode ser até um char errado, mas ela digitou apenas um char.*** Do try cai no if que pega o string que contém todas as opções: “+-\*/=T” mando procurar na string usando o método indexOf para achar a opção que a pessoa digitou; se der -1, lança a exceção opção inválida, é porque não achou. Depois do erro pede o continue do loop anterior. ( pede para a pessoa digitar a opção da lista). Digamos que a pessoa digitou uma opção válida, o indexOf vai achar a posição desse char, o programa continua.

***O programa declara uma variável valor do tipo double a qual inicia com zero.*** Verifica se a opção da operação aritmética for válida (+-\*/), uso o método indexOf para achar a posição, não vai dar -1. Eu peço para digitar um valor, pois o primeiro valor é o zero, o segundo é o que o usuário digitou. Entro no try, para conseguir a digitação do valor declaro teclado.getUmDouble. Se por um acaso o usuário lançar algo que não seja número, o método GetUmDouble gera exceção de valor inválido (cai no catch) e continue e a repetição começa pedindo as opções. Digamos que a pessoa digitou um valor que era um número e foi guardado na variável valor. Eu salto uma linha, verifico se a opção for barra e valor for zero, dou a exceção de valor inválido e dou o continue para voltar do início onde tem a linha das opções.

***Digamos que o usuário passou um valor certo, em seguida eu crio um objeto servidor.receba*** que instancia um novo PedidoDeOperação que tem como parâmetro opção e valor passando para o construtor da classe PedidoDeOperação. O programa do cliente vai enviar um novo PedidoDeOperação, e faz o servidor receber. Depois, temos a possibilidade da pessoa não ter escolhido uma operação, vou verificar se a opção é um igual, caio no elseif. Dentro do else if eu instancio um novo PedidoDeResultado que será passado para o servidor.receba: eu envio pro servidor um pedido de resultado; o método receba na verdade envia, pois o parceiro servidor recebe. Agora, eu declaro um comunicado que recebe nulo. Depois entro no loop do espiando o comunicado que vem (declaro o objeto comunicado receber o servidor.espie e convertido a comunicado.

***Enquanto o comunicado não for uma instância de resultado , eu fico esperando o resultado.*** Na hora que espera o resultado, pode acontecer de vir um PedidoDeDesligamento. Eu espio, se for resultado trato no cliente, se for pedido de desligamento a tratadoradecomunicadodedesligamento trata.

**Se vier uma instância de resultado por comunicado; o loop para, o meu objeto resultado da classe resultado recebe um servidor.envie** que será convertido para Resultado. Isto é, o meu programa recebe o resultado chamando o servidor.envie (se eu falo pro servidor enviar é porque o cliente vai receber). O método envie ele sempre retornar um comunicado por isso que converte a resultado. E dá a mensagem de resultado atual: +resultado.getValorResultante () e um pulador de linha. (Pula 2 linhas). E a main fica na repetição de mostrar as opções.

**O loop fica até quando o usuário escolher a opção T é porque quer terminar.** Todas as coisas que estão no while estão dentro do try. Se der erro dentro do try (na hora de pedir resultado, pedir operação, na hora de interagir com o servidor), caio no catch e escrevo na tela:

Erro de comunicação com o servidor.

Tente novamente !

Caso o erro persista, termine o programa!

E volte a tentar mais tarde!

**Quando o usuário escolher T, ele quer sair**. Após o catch entro dentro do while no try, onde o servidor.receba tem como parâmetro um novo PedidoParaSair (); para que no servidor ele tira o cliente da lista de clientes atendidos. Depois de mandar pro servidor, o programa escreve a mensagem de obrigado por usar este programa, executa system.exit (0). O system.exit significa que eu quero parar o programa, mesmo com o fim da main. Preciso colocar esse comando porque senão a main vai acabar, mas a TratadoraDeComunicadoDeDesligamento iria continuar rodando e o programa não acaba O system.exit ele para todas as tarefas e do programa inteiro. O zero significa término normal; o valor é para informar que ele está terminando e como, se for outro valor seria término anormal, com erros.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**5) Fazedor de Continhas\_Implementação das classes específicas do servidor (parte1).mp4**

**AULA 5 – Fazedor de Continhas: Implementação das classes específicas do servidor (parte 1)**

Hoje o propósito da videoaula **é discutir a infra-estrutura do programa Servidor** que faz parte da arquitetura cliente – servidor que nós estamos estudando, que é o fazedor de Continhas. Tem um conjunto de classes comuns nas implementações do Cliente e do Servidor. Agora vamos focar nas classes que fazem parte exclusivamente da implementação do programa Servidor. Vai começar a explicar sobre as ações que são realizadas no início da main do servidor, tem o propósito de iniciação.

Temos que :

**5.1. Como vai funcionar/ se monta a classe Servidor**

**SERVIDOR**

AÇÃO DE INICIAÇÃO QUE DEVE SER REALIZADA NO **SERVIDOR:**

**ESSAS AÇÕES SÃO REALIZADAS NO INÍCIO DA MAIN DO SERVIDOR**

1. ***Checar os parâmetros vindos para main em args*** para permitir que o usuário indique a porta adotada para aceitar conexões dos Clientes, sinalizando eventuais erros, ou caso nenhum parâmetro venha para a main em args, adotar um porta padrão aceitar conexões dos Clientes;
2. ***Declarar e instanciar a estrutura de dados (ArrayList) “usuários ”*** para armazenar objetos da classe Parceiro onde estarão todos os acessos aos Clientes conectados (que são os parceiros do Servidor); tais acessos serão usados ao longo de todo o programa para se comunicar esses Clientes;
3. “***Startar” uma AceitadoraDeConexao***, fornecendo a ela a porta, na qual conexões serão aceitas, e, naturalmente, compartilhando com ela a estrutura de dados “usuarios” que ela compartilhará com cada SupervisoraDeConexao que ela “startar” , por ocasião de cada conexão que ela aceitar.

-----------------------------------------------------------------------------

**TIPOS DE TAREFAS SIMULTÂNEAS QUE DEVEM SER REALIZADAS NO SERVIDOR**

**APÓS A REALIZAÇÃO DAS AÇÕES DE INICIAÇÃO LISTADAS ACIMA:**

**ISSO TUDO É REALIZADO NO FINAL DA MAIN DO SERVIDOR, QUE FICA EXECUTANDO JUNTO COM AS THREADS “STARTADAS” APÓS EXECUTAR AS TAREFAS DE INICIAÇÃO (item 1)**

1. ***CONTINUAMENTE interagir com o usuário administrador***, avisando a ele que o Servidor está ativo, informando a ele que ele pode desativar o servidor, digitando o comando “desativar”, aguardando que algo seja digitado, informando que o oque foi digitado está incorreto, caso, de fato, estiver, ou, tendo sido digitado de fato o comando “desativar”, enviando a todos os Parceiros cliente conectados, ou seja, presentes na estrutura de dados “usuarios”, um ComunicadoDeDesligamento;

**ISSO TUDO É REALIZADO PELA THREAD ACEITADORADECONEXAO (item 2)**

1. ***CONTINUAMENTE aceitar novas conexões com os programas Cliente que pedirem, “startando” uma SupervisoraDeConexao*** para supervisionar, ou seja, atender qualquer solicitação recebida via a conexão supervisionada e, naturalmente, compartilhando com ela a estrutura de dados “usuarios”;

**ISSO TUDO É REALIZADO POR CADA THREAD SUPERVISORADECONEXAO STARTADA**

1. **INICIALMENTE, instancia um transmissor (ObjectOutputStream) e um receptor (ObjectInputStream) associados à Conexao** que ela supervisiona, para com tudo isso (conexão, transmissor e receptor), instanciar um parceiro cliente, que será usado ao longo de toda sua execução para se comunicar com o programa Cliente, além de incluí-lo na estrutura de dados “usuarios” (para que aquele Parceiro cliente, como qualquer outro presente na estrutura de dados “usuarios”, possa receber Comunicados do Servidor destinados a todos os clientes conectados , o que acontece quando o Servidor recebe comando “desativar”); após tudo isso, passa a CONTINUAMENTE receber o que foi enviado via seu Parceiro cliente, de fomra a atender o que for solicitado e enviando um Resultado ao programa Cliente solicitante, no caso da solicitação ser do tipo PedidoDeResultado, ou removendo seu Parceiro cliente da estrutura de dados “usuários” e terminando sua execução, no caso da solicitação ser do tipo PedidoParaSair.

-----------------------------------------------------------------------------

**5.2.AÇÃO 1 DE INICIALIZAÇÃO**

**Nesse momento em que ocorre as iniciações, é importante saber que acontece apenas a execução exclusiva da main.**  Na ação 1, temos que diz respeito a checagem dos parâmetros que vieram para a main dentro do vetor args. Nós conhecemos o vetor args desde a primeira aula da nossa disciplina, sempre esteve presente em qualquer programa feito anteriormente, mas não sabia a sua utilidade; a sua utilidade ficou explicada na aula da implementação das classes do Cliente. Os parâmetros servem para que o usuário administrador do servidor possa indicar qual porta quer que o servidor adote para aceitar conexões do cliente, espera que se tenha 0 ou 1 parâmetro no vetor args.Se tiver 2 ou mais informações está errado; se tiver erros eles serão sinalizados. Se tiver 1 informação no vetor args, é porque o usuário indicou a porta para atender as conexões dos clientes; seria a única informação mesmo que não fosse numérica ou porta ocupada; então ainda teríamos sinalização de erros. Se a informação passada estiver tudo ok, será passado sem erros; senão tiver informação, é adotado o padrão de dentro do servidor.

**Lembre-se que no programa cliente, para o usuário passar os parâmetros aos programas ao args, poderia ser usado a prompt de comando do sistema operacional**. Quer mostrar uma alternativa; a ideia seria criar atalhos de servidor e cliente (arquivos java) na área de trabalho, menu iniciar. Uma vez criados, se eu der um duplo clique pro atalho pro servidor, ele roda e aparece:

**Se eu fizer o mesmo com o prompt do cliente:**

**5.2.1.PROMPT DE COMANDO SERVIDOR:**

O servidor está ativo! Para desativa-lo

Use o comando “desativar”

**5.2.2.PROMPT DE COMANDO CLIENTE:**

Sua opção (+,-, \*, /, =, [T]erminar)? +

Valor? 8

Sua opção (+,-, \*, /, =, [T]erminar)? =

Resultado atual : 8.0

Sua opção (+,-, \*, /, =, [T]erminar)? /

Valor: 4

Sua opção (+,-, \*, /, =, [T]erminar)? =

Resultado atual : 2.0

Sua opção (+,-, \*, /, =, [T]erminar)?

**O programa cliente está funcionando e não foi acionado nem dentro do ambiente de desenvolvimento e nem digitando o prompt de comando;** foram acionados pelos links que eu criei. Como são esses links e como eu crio?

**Fechou os programas no X pela janela tanto do cliente como do servidor.**

**Para criar os links é muito simples**, é só a gente clicar onde quer criar o link, por exemplo na área de trabalho. Com o botão direito do mouse, abre a guia Exibir, Classificar por, Atualizar, Open Folder as IntelliJ IDEA Community Edition Project, Open Folder as PyCharm Community Edition Project, Open with Code, Configurações Gráficas Intel, Painel de Controle do NVIDIA, novo, configurações de exibição, personalizar. Depois do menu, clica em novo, atalho. Atalho e link é a mesma coisa. Ele criou um atalho com nome de “novo atalho” e apareceu a janela de criar atalho que é para eu digitar o local do item que eu quero copiar, ou seja, qual é o comando que eu daria no prompt de comando para eu executar o que quero que seja executado quando clicar no atalho. Logo, o atalho fica associado a um comando.

**Para rodar os programas feito em java, a gente deve escrever java Servidor ou então java Cliente;** ele já vai colocar os complementos do comando (caminho do item, parâmetros) de forma que ele receba lá no vetor args da main. Então, se eu fosse escrever : java Servidor (deixaria sem parâmetro e ele entraria na porta padrão); ou poderia escrever java Servidor 4000 (que pararia na main do servidor), então ele usou como nome: java Servidor 4000. Dou avançar. Ele vai perguntar o nome do atalho: Serv, clica em concluir. Uma vez concluído, aparece o ataho serv na área de trabalho.

**Embora o meu atalho Serv.java esteja criado ele não está pronto, se der um duplo clique ele fecha automaticamente.** Ele não está funcionando porque falta um detalhe. Eu clico no Serv.exe com o botão direito do mouse e no menu eu escolho Propriedades (17min45s). E nas propriedades, vocês vão ver que tem destino: é o comando que eu escrevi que poderia ser escrito no prompt de comando, porém eu escrevi java Servidor 4000. Notem que tem bem mais do que java; porque ele escreveu o local onde está o java. O java está no C:\Program Files\Java\ jdk-13.0.1\bin

**Nem sempre ele seria capaz de colocar essa pasta; é a configuração do sistema operacional; senão estivesse configurado,** o caminho antes de bin não estaria escrito e eu teria que puxar o caminho de onde ele está; eu procuro no explorador de arquivos, localizo o java, copio do caminho e colo no destino das propriedades. O iniciar em está preenchido em automático, mas está errado, aparece: C:\Program Files\Java\ jdk-13.0.1\bin isso pode acontecer em atalhos ou em qualquer outra situação.

**Em Iniciar em deveria estar escrito onde está instalado o programa Servidor, onde estão compilados e instalados todos os ponto class do programa servidor?**  Em que pasta está o programa Servidor? Então, eu vou no C:\ Fazedor de Continhas Cliente-Servidor\Servidor

**É na pasta servidor que estão os .class, é o resultado da compilação das minhas classes que constituem o servidor.** Copio o caminho do explorador de arquivos e colo em iniciar em. Para cada arquivo.java tem um .class correspondente. Clico em Aplicar, ok. O link já foi atualizado. Observem que se eu der um duplo clique em Serv 4000 ele abre normalmente com a escrita de prompt:

O servidor esta ativo! Para desativa-lo, use o comando “desativar” >

**Uma outra coisa que ele quer nos mostrar é sobre o pequeno detalhe. O atalho do cliente está feito e funcionando. Como o atalho do cliente foi criado?** Da mesma forma que o atalho serv; mas tem uma pequena observação aqui: quando foi mostrado como se ativa um programa pelo prompt de comando, eu falei que na hora de ativar o cliente eu teria que escrever na frente de java cliente ou nada para opções padrão, ou a máquina onde está rodando o servidor, ou poderia escrever a máquina e a porta onde rodam o servidor. E lá na ocasião, escrevi localhost como máquina local e por está rodando cliente e servidor, por está com uma máquina só não tem problema não colocar o nome da máquina. Mas se estivesse em máquinas separadas, teria que colocar o nome da máquina ou o ip, e localhost é o nome da máquina. E no atalho cliente está escrito o ip.

**Ao selecionar o atalho de Cliente, clico com o botão direito, propriedades, Destino.**

O destino ficou sendo “C:\Program Files\Java\ jdk-13.0.1\bin\java.exe” Cliente 192.168.0.177 4000 tem o local de onde ficou o java, o cliente, ip e porta.

E no iniciar em: C:\ Fazedor de Continhas Cliente-Servidor\Cliente é a pasta onde fica o cliente, os programas e classes do cliente.

**Tanto os atalhos Servidor, Serv, e Cliente funcionaram ao clicar com o botão esquerdo 2x**.

Vamos voltar as ações de inicialização.

**5.3.AÇÃO 2 DE INICIALIZAÇÃO**

**Temos que no início da main do servidor, após fazer a ação 1 devemos fazer a ação 2.** Na ação 2 temos que fazer o seguinte: declarar e instanciar a estrutura dos dados, que é o ArrayList mas poderia ser uma estrutura de armazenamento qualquer. Essa estrutura vai se chamar usuários (objeto) da classe ArrayList; para armazenar objetos da classe Parceiro onde vão estar todos os acessos aos clientes conectados. Ou seja, os parceiros do servidor são os clientes conectados. Teremos muito objetos da classe parceiro dentro do arraylist, cada um é um acesso ao cliente conectado ao servidor. Os acessos serão usados durante todo o programa, para se comunicar com os clientes conectados. A classe Parceiro era uma das classes comuns aos programas Cliente e Servidor. A classe Parceiro tinha entre os seus atributos: conexão (Socket), receptor (ObjectInputStream) e transmissor (ObjectOutputStream). A classe parceiro tinham os métodos envia, receba e espie. Esses métodos serviam para que o servidor pedisse pro cliente enviar algo, ou para que no cliente o servidor mandasse alguma coisa; eu chamava o método envie.

**Para usar o método receba, eu pedia dentro do servidor para que o cliente recebesse algo**, ou para que dentro do cliente o servidor recebesse alguma coisa. As classes de Parceiro faziam com que tivesse fluxo de informações com o parceiro conectado.

**E no servidor nós vamos ter vários clientes conectados. No cliente tinha um objeto da classe Parceiro que era o servidor, porque o cliente somente se comunicava com um servidor.** Mas, na classe Servidor, um servidor pode ter 1 ou mais clientes conectados a ele; pois cada usuário pode usar um ou mais programas que ele queira na máquina dele. Você pode colocar um programa cliente para rodar da sua casa, eu posso colocar um programa cliente pra rodar na minha casa. O servidor vai se conectar com o meu e o seu programa cliente para se comunicar. E os programas cliente pode ser de 1 a 100 programas cliente para o mesmo servidor. Então, o servidor pode dar conta de atender um monte de clientes conectados. Logo, no servidor vai ter muitos objetos da classe parceiro e vão estar todos armazenados no arraylist.

**5.4.AÇÃO 3 DE INICIALIZAÇÃO**

**Para a ação de iniciação número 3,** temos que declarar, instanciar e startar uma AceitadoraDeConexão . A AceitadoraDeConexao é uma classe que existe no Servidor que herda de thread, portanto é uma tarefa; que terá a missão de aceitar conexões dos programas clientes. O construtor dessa classe tem dois parâmetros, então a main vai passar pro construtor dela a porta e a estrutura de dados “usuários”. Ou seja, a main vai compartilhar com a classe AceitadoraDeConexao a estrutura de dados “usuários”. E ela, a cada conexão aceita, vai declarar, instanciar e startar uma SupervisoraDeConexao que é uma classe que herda de thread e tem a missão de supervisionar uma conexão, cuidar dessa conexão. E a outra missão da AceitadoraDeConexão é que a cada conexão aceita, startar a SupervisoraDeConexao.

**Logo, teremos rodando a main, uma AceitadoraDeConexao e várias SupervisoraDeConexao;** tudo ao mesmo tempo.

**5.5. AÇÕES DE TAREFAS SIMULTÂNEAS**

(Parei em 38min00s).

**Vamos ter ações que serão realizadas no final da main do servidor e temos tarefas simultâneas sendo executadas; após a ação 3 onde starta a AceitadoraDeConexão**, vai estar rodando: main, AceitadoraDeConexão e n SupervisoraDeConexão rodando tudo ao mesmo tempo. Então vão ser várias tarefas em execução simultânea. No início da main do servidor o que ainda faz parte desse início é declarar, instanciar e startar uma AceitadoraDeConexao; o construtor dela vai receber a porta das conexões dos clientes onde vão ser aceitas e a estrutura de dados usuários.

**A AceitadoraDeConexao compartilha a estrutura de dados usuários** com cada SupervisoraDeConexao que ela startar. Então, aceitou uma conexão, starta uma supervisora e vai passar pro construtor de supervisora a estrutura de dados de usuário.

Depois das ações que ocorrem no início da main do servidor, **entramos na etapa de execução de tarefas simultâneas.** As tarefas vão estar em execução simultânea e a main vai ser uma dessas tarefas. O final da main do servidor fica executando junto com os threads que foram startadas após executar as tarefas de iniciação. Tem que ter pelo menos duas tarefas em execução : main do servidor e AceitadoraDeConexão, mas pode ter mais devido a AceitadoraDeConexoes starta uma SupervisoraDeConexao uma para cada conexão de cliente que aceitar. Se a Aceitadora não aceitou nenhuma conexão, então ela ainda não colocou nenhuma supervisora para rodar.

**5.5.1.NA AÇÃO 1 DAS TAREFAS SIMULTÂNEAS, TEMOS QUE**

**Vamos ver o papel da main que fica junto com as threads**. Espera-se que a main continuamente (haverá uma repetição) interaja com o usuário administrador avisando que o servidor está ativo e informa que pode desativar o servidor pelo comando desativar e aguarde que o usuário digite.

**Quando a digitação acontecer, vai ver se o comando é correto; se incorreto dará erro.** Se o comando for correto, a main envia para todos os parceiros clientes que estão conectados (estão no usuários) um comunicado de desligamento e em seguida, a main do servidor termina o programa e todas as threads são finalizadas após o envio do comunicado.

**5.5.2.NA AÇÃO 2 DE TAREFAS SIMULTÂNEAS, TEMOS QUE:**

**A ação 2 é realizada pela thread aceitadoraDeConexao e essa thread tem o método run que vai estar executando e tem um loop.** Para que continuamente, a aceitadoraDeConexao aceite novas conexões com os programas que pedirem. Assim que uma conexão for aceita, ela vai startar uma thread do tipo SupervisoraDeConexao para atender qualquer solicitação recebida pela conexão recém-aceita e a aceitadora compartilha com a supervisora a estrutura de dados “usuarios”.

**A conexão é passada como parâmetro para o construtor de SupervisoraDeConexao**. A AceitadoraDeConexao aceitou a conexão, ela tem a conexão e vai querer startar uma supervisora para supervisionar a conexão. Logo, a AceitadoraDeConexao vai declarar, instanciar a supervisora antes de startar, E na hora de instanciar a supervisora, ela vai passar como parâmetro ao construtor da supervisora a conexão que ela acabou de aceitar e passa a estrutura de dados usuários (o usuários é compartilhado com a supervisora, que é onde tem todos os clientes conectados).

**Observe que a AceitadoraDeConexao não faz muita cois**a. Pelo fato da AceitadoraDeConexoes estar em loop, após startar a SupervisoraDeConexao, a aceitadora volta e passa a aguardar novas conexões para aceitar; voltando ao início do loop. (Parei em 53min42s). É bom que a aceitadora seja curta e rápida, pois não pode demorar muito para voltar a aceitar outra conexão, pode ser que o segundo cliente peça conexão e a thread esteja ocupada na conexão anterior.

**5.5.3 NA AÇÃO 3 DE TAREFAS SIMULTÂNEAS, TEMOS QUE:**

**É um tipo de tarefa que é realizado por cada thread SupervisoraDeConexao startada.** Várias threads desse tipo podem ser startadas pela AceitadoraDeConexão e cada thread do tipo SupervisoraDeConexao vai fazer o seguinte:

**Inicialmente, a SupervisoraDeConexao vai declarar e instanciar vários objetos, tais como**: um transmissor que é do tipo ObjectOutputStream, um receptor do tipo ObjectInputStream. Os dois objetos vão estar associados a conexão que ela supervisiona, que é a conexão que foi recebida como parâmetro de construtor da thread SupervisoraDeConexao mandada pela thread aceitadora.

**Com tudo isso, a thread supervisora vai declarar e instanciar um Parceiro cliente que vai ser usado ao longo de toda a execução da thread para se comunicar com o cliente além de incluí-lo na estrutura de dados usuários.** Então a nossa thread supervisora recebeu a conexão, com a conexão criou transmissor, receptor e o parceiro cliente; e com esse parceiro vai incluir no array de usuários. O parceiro cliente que acabou de instanciar, possa receber comunicados do servidor destinados a todos os clientes.

**Logo, sempre que o servidor precisar mandar comunicados que são destinados a todos os clientes, ele vai usar a estrutura de dados usuários**, pois nela estão todos os parceiros clientes que correspondem a todos os clientes que estão conectados. Então, o servidor usa a estrutura usuários. A estrutura de dados deve conter todos os clientes conectados.

**Quando o servidor precisa mandar comunicados a todos os clientes?** Isso acontece quando o servidor recebe o comando desativar pelo usuário administrador que digita esse comando. O servidor vai precisar mandar para todos os clientes conectados e precisa se comunicar com os clientes da estrutura de usuários, e manda o ComunicadoDeDesligamento.

**A SupervisoraDeConexao fazendo as coisas iniciais, após isso ela passa CONTINUAMENTE a receber o que foi enviado do seu Parceiro cliente;** para atender o que foi solicitado. Logo, se foi solicitado uma operação, ela vai fazer a operação requerida (adição, multiplicação, divisão, subtração); se foi solicitado um PedidodeResultado, ela vai enviar o resultado ao programa Cliente solicitante, ou a supervisora, eventualmente, remover o Parceiro cliente da estrutura de dados se ele fizer o PedidoParaSair e essa thread termina a sua execução. Tem uma thread supervisora para cada parceiro cliente.

**Antes de terminar a sua execução, a SupervisoraDeConexao executa um método que existe no Parceiro cliente, que se chama adeus.** O método adeus faz a desconexão do transmissor, receptor, conexão; desfaz todas as conexões. Então, quando a SupervisoraDeConexao receber um PedidoParaSair, ela remove o seu Parceiro cliente da estrutura de usuário, executa o método adeus do Parceiro cliente. E após, a thread termina a sua execução. ( A thread Supervisora morre).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**6) Fazedor de Continhas\_implementação das classes específicas do Servidor (parte2).mp4**

**AULA 6 – Fazedor de Continhas: Implementação das classes específicas do Servidor (parte 2)**

É o **último vídeo da série que aborda o Fazedor de Continhas** do sistema Cliente-Servidor.

**6.1.Arquivos usados:**

**6.1.1.Arquivo: Servidor.java**

import java.util.\*;

public class Servidor

{

public static String PORTA\_PADRAO = "3000";

public static void main (String[] args)

{

if (args.length>1)

{

System.err.println ("Uso esperado: java Servidor [PORTA]\n");

return;

}

String porta=Servidor.PORTA\_PADRAO;

if (args.length==1)

porta = args[0];

ArrayList<Parceiro> usuarios =

new ArrayList<Parceiro> ();

AceitadoraDeConexao aceitadoraDeConexao=null;

try

{

aceitadoraDeConexao =

new AceitadoraDeConexao (porta, usuarios);

aceitadoraDeConexao.start();

}

catch (Exception erro)

{

System.err.println ("Escolha uma porta apropriada e liberada para uso!\n");

return;

}

for(;;)

{

System.out.println ("O servidor esta ativo! Para desativa-lo,");

System.out.println ("use o comando \"desativar\"\n");

System.out.print ("> ");

String comando=null;

try

{

comando = Teclado.getUmString();

}

catch (Exception erro)

{}

if (comando.toLowerCase().equals("desativar"))

{

synchronized (usuarios)

{

ComunicadoDeDesligamento comunicadoDeDesligamento =

new ComunicadoDeDesligamento ();

for (Parceiro usuario:usuarios)

{

try

{

usuario.receba (comunicadoDeDesligamento);

usuario.adeus ();

}

catch (Exception erro)

{}

}

}

System.out.println ("O servidor foi desativado!\n");

System.exit(0);

}

else

System.err.println ("Comando invalido!\n");

}

}

}

++++++++++++++++++++++++++++++++++++

**6.1.2.Arquivo: AceitadoraDeConexao.java**

import java.net.\*;

import java.util.\*;

public class AceitadoraDeConexao extends Thread

{

private ServerSocket pedido;

private ArrayList<Parceiro> usuarios;

public AceitadoraDeConexao

(String porta, ArrayList<Parceiro> usuarios)

throws Exception

{

if (porta==null)

throw new Exception ("Porta ausente");

try

{

this.pedido =

new ServerSocket (Integer.parseInt(porta));

}

catch (Exception erro)

{

throw new Exception ("Porta invalida");

}

if (usuarios==null)

throw new Exception ("Usuarios ausentes");

this.usuarios = usuarios;

}

public void run ()

{

for(;;)

{

Socket conexao=null;

try

{

conexao = this.pedido.accept();

}

catch (Exception erro)

{

continue;

}

SupervisoraDeConexao supervisoraDeConexao=null;

try

{

supervisoraDeConexao =

new SupervisoraDeConexao (conexao, usuarios);

}

catch (Exception erro)

{} // sei que passei parametros corretos para o construtor

supervisoraDeConexao.start();

}

}

}

++++++++++++++++++++++++++++++++++++++++++++

**6.1.3.Arquivo: SupervisoraDeConexao.java**

import java.io.\*;

import java.net.\*;

import java.util.\*;

public class SupervisoraDeConexao extends Thread

{

private double valor=0;

private Parceiro usuario;

private Socket conexao;

private ArrayList<Parceiro> usuarios;

public SupervisoraDeConexao

(Socket conexao, ArrayList<Parceiro> usuarios)

throws Exception

{

if (conexao==null)

throw new Exception ("Conexao ausente");

if (usuarios==null)

throw new Exception ("Usuarios ausentes");

this.conexao = conexao;

this.usuarios = usuarios;

}

public void run ()

{

ObjectOutputStream transmissor;

try

{

transmissor =

new ObjectOutputStream(

this.conexao.getOutputStream());

}

catch (Exception erro)

{

return;

}

ObjectInputStream receptor=null;

try

{

receptor=

new ObjectInputStream(

this.conexao.getInputStream());

}

catch (Exception err0)

{

try

{

transmissor.close();

}

catch (Exception falha)

{} // so tentando fechar antes de acabar a thread

return;

}

try

{

this.usuario =

new Parceiro (this.conexao,

receptor,

transmissor);

}

catch (Exception erro)

{} // sei que passei os parametros corretos

try

{

synchronized (this.usuarios)

{

this.usuarios.add (this.usuario);

}

for(;;)

{

Comunicado comunicado = this.usuario.envie ();

if (comunicado==null)

return;

else if (comunicado instanceof PedidoDeOperacao)

{

PedidoDeOperacao pedidoDeOperacao = (PedidoDeOperacao)comunicado;

switch (pedidoDeOperacao.getOperacao())

{

case '+':

this.valor += pedidoDeOperacao.getValor();

break;

case '-':

this.valor -= pedidoDeOperacao.getValor();

break;

case '\*':

this.valor \*= pedidoDeOperacao.getValor();

break;

case '/':

this.valor /= pedidoDeOperacao.getValor();

}

}

else if (comunicado instanceof PedidoDeResultado)

{

this.usuario.receba (new Resultado (this.valor));

}

else if (comunicado instanceof PedidoParaSair)

{

synchronized (this.usuarios)

{

this.usuarios.remove (this.usuario);

}

this.usuario.adeus();

}

}

}

catch (Exception erro)

{

try

{

transmissor.close ();

receptor .close ();

}

catch (Exception falha)

{} // so tentando fechar antes de acabar a thread

return;

}

}

}

++++++++++++++++++++++++++++++++++++++++  
 Nesse **vídeo deseja mostrar a implementar as classes que são específicas para o servidor**. Para relembrar as ações da aula passada (tanto as ações de inicialização como as ações das tarefas simultâneas).

**Após a leitura da ação 1 de inicialização, abre o código de Servidor.java para avaliar a main.**

Quando a quantidade de parâmetros for maior do que 1, sinalizo o erro e o uso esperado é java servidor e opcionalmente uma porta. (É o if (args.length>1). Se der erro, dá o return para terminar a main dentro do if. Senão entrar nesse if, eu declaro uma String chamada porta que recebe Servidor.porta\_padrão.

**Antes dos ifs, na declaração de public static String PORTA\_PADRAO= “3000”**; colocar a palavra final antes de String para sinalizar que é uma constante. O nome da constante é porta padrão.

**Se o parâmetro for realmente 1, (é o if (args.length==1)** a porta ficará na posição zero do args; ou seja, se houver 1 parâmetro será guardado em porta, substituindo a porta padrão. É nessa parte que fica a indicação da porta.

**No item 2/ ação 2 de ação de inicialização, vamos verificar no código.** Após o comprimento de args ser 1, declaramos o ArrayList que guarda parceiros com nome usuários receber uma nova instanciação ArrayList <Parceiro>();

**Na ação 3 de iniciação, vamos verificar no código.** Após declarar o ArrayList, declaro uma AceitadoraDeConexao como um objeto aceitadoraDeConexao que recebe nulo. Entro no sistema try-catch. No try, eu faço a aceitadoraDeConexao receber uma instanciação de AceitadoraDeConexao e passo ao construtor a porta e usuários. Após, (na linha debaixo), eu chamo o método start para a aceitadoraDeConexao. No catch é para dar erro de escolher a porta apropriada e dar um return.

**Na parte de tarefas simultâneas, temos três ações.**

**A primeira tarefa simultânea é feita pela própria main na sua parte final do servidor;** fica executando junto com as threads startadas, No código, temos um for infinito que implementa um loop infinito. E nesse loop acontece de eu printar as seguintes coisas:

“O servidor está ativo! Para desativa-lo,”

“use o comando \”desativar\” \n” // quando tem barra -aspa tem o sentido de abrir e fechar uma string. O que faz parte da string: use o comando “ desativar” o \n para pular uma linha e fecho em “. (após o \n), então pula 2 linhas.

“>” // indicação de que o servidor está pronto para ler uma indicação do comando

**Depois, cria um objeto comando que recebe nulo.** Após, entra num try catch, onde no try o comando recebe a classe Teclado com o método getUmString atrelado; e no catch vazio entre colchetes. O teclado com getUmString é para captar uma string que o usuário digitar no teclado e guardo em comando.

Depois de pegar o que a pessoa digitou, vou validar se realmente é o comando desativar ou não.

Se o comando em letra minúscula for igual a desativar (é o if (comando.toLowerCase ().equals (“desativar”)). Eu uso equals para comparar conteúdos entre objetos, pois o == compara endereços de memória , pois todo objeto é um ponteiro. Eu quero comparar aquilo que ele aponta, que por isso que se usa o equals. O toLowerCase é um método para transformar letras maiúsculas em minúsculas, não importe como o usuário digite.

**Eu entro no if e faço o parâmetro usuários ser synchronized** , significa que eu quero acesso exclusivo a usuários sem que nenhuma outra thread use ao mesmo tempo. Para os usuários, eu declaro um objeto comunicadoDeDesligamento do tipo/ classe ComunicadoDeDesligamento que é instanciado como novo ComunicadoDeDesligamento ().

**Após eu faço o for com sintaxe diferente pelos dois pontos.** A gente chama como foreach ou para cada. Para cada Parceiro usuário dentro dos usuários. Eu entro no sistema try – catch. No try eu faço o usuário receber um comunicadoDeDesligamento e na linha seguinte o usuário estará atrelado ao método adeus (). No catch é um par de colchetes nulos. O método adeus na classe Parceiro usa o método close para desconectar os seus objetos: transmissor, receptor e conexão.

**O foreach é o segundo for. O primeiro for é o for (;;).** A main executa com aceitadoraDeConexao e possivelmente com várias supervisoraDeConexao.

(Parei em 20min05s).

Agora vamos falar a respeito do **segundo tipo de tarefa simultânea** que é realizada no servidor.

**É realizado pela thread aceitadoraDeConexão. Ele vai substituir por um texto mais abrangente, atualizado. Logo:**

**ISSO TUDO É REALIZADO PELA THREAD aceitadoraDeConexao**

2) ***INICIALMENTE, declara e instancia um objeto da classe ServerSocket***, que é o que se usa para aceitar pedidos de conexões (ele tem um método chamado accept), além de guardar em um atributo (para poder ser usado ao longo de toda a classe, em todos os métodos) a estrutura de dados “usuários” que foi compartilhada pela main ao instanciar esta thread; depois, passa a CONTINUAMENTE aceitar novas conexões com os programas Cliente que pedirem, declarando, instanciando e “startando” uma SupervisoraDeConexao para supervisionar, ou seja, atender qualquer solicitação recebida via a conexão supervisionada e, naturalmente compartilhando com ela a estrutura de dados “usuários”;

**Após a leitura do item 2, o ServerSocket precisa receber a porta que vai ser usada para aceitar os pedidos de conexão do Cliente**. O objeto chamante (da classe ServerSocket) possui o método accept; esse método é que é capaz de aceitar pedidos de conexões.

**Na classe AceitadoraDeConexão, vemos o import das bibliotecas net e útil; o import serve para declarar objetos das classes ServerSocket, Socket que estão contidas nas bibliotecas citadas anteriormente**. Na biblioteca java.util tem o ArrayList. Começa a classe AceitadoraDeConexao herda de Thread. Tem dois atributos privativos; o primeiro é pedido da classe ServerSocket. O segundo é usuários da classe/tipo ArrayList que armazena parceiros. Os parceiros do servidor são os clientes que pedem conexões e tem os acessos guardados no arraylist dentro de um objeto da classe Parceiro.

**Lembrando que, Parceiro tem conexão (do tipo Socket), receptor (tipo ObjectInputStream) e transmissor (do tipo ObjectOutputStream).**

**Após, tem o construtor da nossa classe que recebe a String porta, ArrayList <Parceiro> usuários**. A porta pode ter sido indicada pelo usuário administrador quando ele acionou o programa, pode ter sido pega pelo vetor args. Porém se o administrador não indicou a porta, essa porta pode ser a padrão. O fato é que a main passou a porta e a estrutura de dados usuários que é um arraylist de parceiros. O meu construtor verifica, no primeiro momento, se realmente foi passado a porta , senão dá a exceção de porta ausente.

**Caso a exceção não tenha acontecido, entro no try e instancio o objeto pedido com new ServerSocket e passo como parâmetro ao ServerSocket** a porta que foi passada na main transformada em int (pelo método do parseInt). O parseInt pode dar erro na transformação para int porque a string porta contenha caracteres que não sejam exclusivamente numéricos e por isso dá erro.

**Outro tipo é quando: a transformação foi tudo ok, e passo a transformação ao ServerSocket , pode ser que a porta que estou passando ao servidor do ServerSocket seja inválida**: negativo (não existe porta negativa) , grande demais (as portas tem um limite de aproximadamente 65000), ocupada (não consigo usar porque está ocupada). Então, se der erro na porta eu lanço a exceção no catch de porta inválida.

**Senão acontecer, vou verificar se foi passada a estrutura de dados usuários; senão foi passada porque deu nulo, lança a exceção de usuários ausentes**. Se foi passado, eu pego os usuários que foram recebidos e guardo no atributo. Acabou o construtor.

**Após o término do construtor, eu tenho a estrutura de dados recebida da main (no começo do programa) e o ServerSocket de pedido com a porta passando como atributo ao construtor.** Os dois atributos estão declarados e instanciados, prontos para serem usados no restante da classe.

**No resto da classe, por ser thread eu tenho o método run que executa toda vez que starta a thread.** No método run tem um loop de for (;;;) para ter a aceitação de novas conexões. Dentro desse loop tem um try-catch. Eu declaro um socket conexão e faço ele receber pedido.accept, ou seja, eu aceito um pedido e gero uma conexão. Para cada conexão que for aceitar, precisa criar uma supervisora. Então depois da instanciação tem o catch de continue. Após o catch, eu tenho a declaração de uma SupervisoraDeConexao valendo nulo; a instanciação da supervisoraDeConexao valendo new SupervisoraDeConexão que passo pro construtor da classe SupervisoraDeConexao a conexão e o usuário. Na aceitadora eu aceito a conexão, essa conexão passo para supervisora e o seu usuário (estrutura de dados). Como no catch sei que não vai dar erro, então eu starto a supervisoraDeConexao. (36min12s).

**O terceiro tipo de tarefa simultânea que acontece no servidor, é importante ressaltar que é um tipo de tarefa que pode ser executado em multiplicidade**: só tenho 1 main e 1 final da main que fica executando com uma aceitadora. Mas posso ter 1 ou várias supervisoras, porque cada conexão que for aceita pela aceitadora vai gerar uma supervisora, porque cada conexão vai ser declarada, instanciada e startada uma supervisora. As supervisoras podem executar em multiplicidade. (Parei em 39min).

**No final da main fica executando simultaneamente com uma aceitadora**, que por sua vez, executa simultaneamente várias supervisoras; pois será executada uma supervisora para cada conexão.

**Após a leitura da ação 3, vamos verificar o código**.

**Nos códigos as bibliotecas usadas de java são io (para usar ObjectOuputStream e ObjectInputStream), net (para usar sockets) e útil (para usar ArrayList).** A classe SupervisoraDeConexao herda de thread, e declara os atributos valor do tipo double que inicialmente vale zero; um atributo usuário do tipo Parceiro ; um atributo conexão do tipo Socket e um atributo usuários do tipo ArrayList que armazena Parceiro. O Parceiro usuário é o cliente que irei supervisionar.

**A main compartilha com a aceitadora e a aceitadora compartilha com a supervisora a estrutura de dados.**

**Faz o método construtor de SupervisoraDeConexão passando como parâmetro o Socket conexão e o ArrayList <Parceiro> usuários** . Vou validar se a conexão é nula, se sim recebe a exceção de conexão ausente e depois se o usuários for nulo também gera exceção de usuários ausentes. Se o construtor realmente recebeu os parâmetros, a conexão recebida como parâmetro e guarda no atributo conexão; de igual modo, ele pega o usuários recebido como parâmetro e guarda no atributo usuário. São os atributos declarados no começo da classe Supervisora.

**Depois do construtor, temos a inicialização da thread com o método run.** O método run, antes da parte do continuamente, tem a declaração do transmissor e dentro do try a instanciação do transmissor que recebe um novo ObjectOutputStream que recebe como parâmetro de construtor a conexão do transmissor que captura o OutputStream (). No catch dá um return.

**Depois, declara o objeto receptor valendo nulo, tenta instanciar o receptor fazendo um novo ObjectInputStream** receber como parâmetro a conexão com o método getInputStream ().

Depois, **declara no catch nulo; após tenta instanciar o objeto usuário receber um novo Parceiro** que tem como parâmetros: conexão, receptor e transmissor e após, um catch vazio.

**Após todos os try-catchs de objetos, vamos requisitar uso exclusivo da estrutura de dados do usuários,** isto é, queremos usar com exclusividade e vamos adicionar a estrutura de dados do usuário aquele usuário que acabou de instanciar como parceiro.

**Agora, dentro de usuário já tem mais um cliente do servidor que esteja armazenado lá dentro;** e quando o servidor quiser mandar uma mensagem/comunicado para todos os clientes, os usuários (todos) vão receber.

**Agora, vamos entrar na parte do continuamente**

**Tem o loop de for (;;) e dentro dele, declaro um objeto comunicado da classe Comunicado e faço o comunicado receber this.usuario.envie,** ou seja, estou pedindo para o cliente que essa supervisora está supervisionando, estou pedindo para o usuário me enviar um comunicado. O comunicado que ele me enviar eu guardo em comunicado.

**Senão veio comunicado nenhum, eu termino a execução do método**. Senão, veio o comunicado, e vou ver o tipo . Se o comunicado é uma instancia da classe PedidoDeOperação, então eu faço a declaração do objeto pedidoDeOperacao da classe PedidoDeOperação e coloquei nesse objeto o comunicado que acabei de receber, visto como um PedidoDeOperação (estou forçando o comunicado a ser visto como algo da classe PedidoDeOperação, posso fazer isso porque testei o tipo da classe usando instanceOf antes). Logo, tenho guardado o pedidoDeOperacao recebido.

Agora, vou pegar o objeto pedidoDeOperacao para o método getOperação (), ou seja, preciso saber qual é a operação requirida. Uso o switch

Se for +, vou pegar o valor e somar o pedidoDeOperacao com getValor e depois break.

Se for -, vou pegar o valor e subtrair o pedidoDeOperacao com getValor e depois break.

Se for \*, vou pegar o valor e mulitplicar o pedidoDeOperacao com getValor e depois break.

Se for /, vou pegar o valor e dividir o pedidoDeOperacao com getValor, não preciso de break porque é o último valor.

**Se o pedido não for de operação, caio no elseif e texto se o comunicado for uma instancia de PedidoDeResultado,** O cliente quer saber quanto que está armazenado no valor nesse momento. Então, se o usuário quer receber o resultado, eu crio uma instancia de Resultado que terá esse valor.

**Se o pedido não for de resultado, é um pedido para sair.** Testo se o comunicado é uma instancia do método PedidoParaSair, peço uso exclusivo do uso da estrutura de armazenamento, e removo dessa estrutura o usuário que fez a solicitação, após isso é feito o método adeus do usuário. O método adeus desconecta desse usuário o receptor, transmissor e conexão através do método close, é melhor explicado na classe Parceiro.

**Logo, a thread supervisora atende aos desejos do usuário que é o cliente**.