ECM251 – Linguagens de Programação I

Aula 20 – L1/1, L2/1 e L3/1

Engenharia da Computação – 3ª série

Cliente-Servidor com Java Sockets
(L1/1, L2/1 e L3/1)

2025

Aula 20 – L1/1, L2/1 e L3/1

Horário

Terça-feira: 2 x 2 aulas/semana

- L1/1 (07h40min-09h20min): *Prof. Calvetti*;
- L1/2 (09h30min-11h10min): *Prof. Calvetti*;
- L2/1 (07h40min-09h20min): *Prof. Menezes*;
- L2/2 (11h20min-13h00min): *Prof. Calvetti*;
- L3/1 (09h30min-11h10min): *Prof. Evandro*;
- L3/2 (11h20min-13h00min): *Prof. Evandro.*

Tópico

- Arquitetura Cliente-Servidor

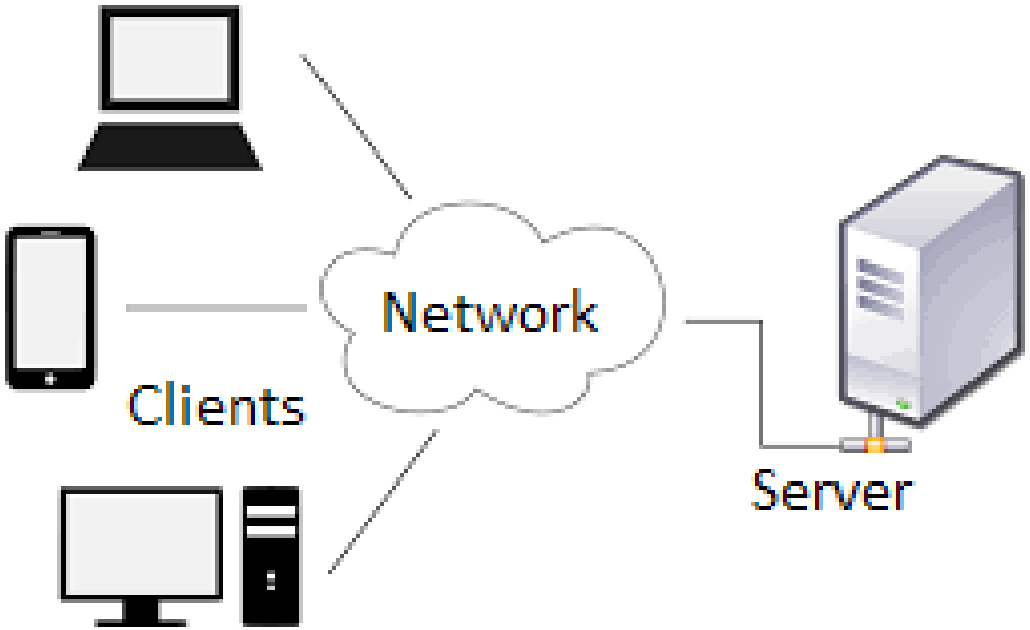
Definição

- Na Engenharia de *Software*, a Arquitetura Cliente-Servidor, *i.e. Client-Server Architecture*, é um modelo fundamental que descreve a distribuição de funções e responsabilidades em um sistema de *software*;
- Nesse modelo, o sistema é dividido em duas partes principais, Cliente e Servidor, que interagem entre si, via rede de comunicação e seus protocolos de comunicação, para realizar tarefas específicas através de transações e interações.

Arquitetura Cliente-Servidor

Definição

- *Client-Server Architecture*:



Definição

- As principais características da Arquitetura Cliente-Servidor são:
  - Cliente:
    - Parte do sistema que solicita serviços ou recursos do servidor;
    - Responsável por apresentar a interface de usuário ao usuário final e coletar entradas;
    - Envia solicitações ao servidor e espera por suas respostas.

Definição

- As principais características da Arquitetura Cliente-Servidor são:
  - Servidor:
    - Parte do sistema que fornece os serviços ou recursos solicitados pelo cliente;
    - Aguarda as solicitações dos clientes, processa essas solicitações e retorna as respostas correspondentes;
    - Possui, geralmente, recursos e informações que os clientes não têm, tornando-o uma fonte de autoridade.

Definição

- As principais características da Arquitetura Cliente-Servidor são:
  - ➢ Comunicação:
    - Entre o cliente e o servidor, geralmente ocorre por meio de **protocolos de rede**, como *HTTP*, *TCP*/*IP*, ou outros protocolos personalizados;
    - O cliente se comunica com o servidor enviando uma solicitação e aguardando uma resposta, podendo a comunicação ser **síncrona**, quando o cliente, após sua solicitação, aguarda por essa resposta, ou **assíncrona**, quando o cliente, após sua solicitação, continua sua execução enquanto aguarda por essa resposta.

Definição

- As principais características da Arquitetura Cliente-Servidor são:
  - ➢ Distribuição de tarefas:
    - Permite a distribuição de tarefas entre o cliente e o servidor de acordo com suas respectivas funções e responsabilidades;
    - Tarefas relacionadas à lógica de negócios, processamento de dados e gerenciamento de recursos geralmente são manipuladas pelo servidor;
    - Tarefas relacionadas à interação com o usuário e apresentação de informações são tratadas pelo cliente.

Definição

- As principais características da Arquitetura Cliente-Servidor são:
  - Escalabilidade:
    - Permite escalabilidade ao adicionar mais clientes e servidores conforme necessário;
    - Pode ser implementada de várias maneiras, *e.g.* como cliente-servidor de três camadas: cliente; servidor de aplicação; e servidor de banco de dados, para sistemas mais complexos.

Definição

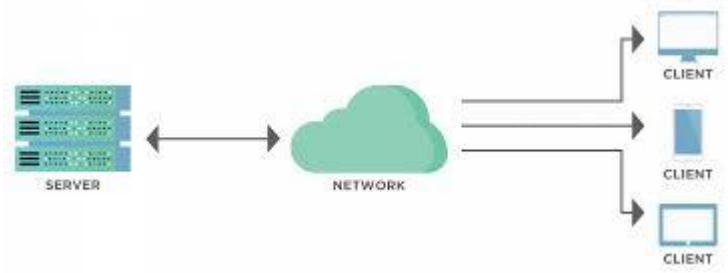
- A arquitetura Cliente-Servidor é amplamente utilizada em muitos tipos de sistemas distribuídos:
  - Aplicativos *Web*, onde o navegador atua como cliente e um servidor *web* fornece conteúdo;
  - Sistemas de banco de dados, onde clientes solicitam dados de um servidor de banco de dados; e
  - Muitos outros cenários em que a comunicação entre diferentes partes é necessária para fornecer funcionalidade e serviços.

Tópico

- Aplicações *Java* para Cliente e Servidor

Definição

- As aplicações *Java* para Cliente e Servidor seguem a arquitetura Cliente-Servidor, composta por:
  - Uma aplicação *Java* Servidor de um lado, responsável por gerenciar as aplicações dos clientes e intermediar a comunicação entre eles pela rede; e
  - Aplicações *Java* Cliente do outro lado, conectadas com o servidor e com os outros clientes através do servidor, utilizando diferentes tipos de dispositivos em rede.



Cliente-Servidor com Java Sockets

Tópico

- *Sockets*

Sockets

Definição

- Recurso do Sistema Operacional – SO que permite que aplicações diferentes, em máquinas diferentes, ou até na mesma máquina, se comuniquem entre si, através de trocas de mensagens em uma rede;
- Permite que uma aplicação servidora se comunique com uma ou várias aplicações clientes e vice-versa;
- A comunicação entre as aplicações ocorre por meio de algum protocolo de comunicação em rede, como o *TCP/IP* ou o *UDP*;
- O *TCP/IP* é um protocolo de transporte que solicita a confirmação da entrega da mensagem transmitida;
- Já o *UDP* não solicita essa confirmação de entrega.

Sockets

Definição

- *Socket*, em Inglês, significa **tomada**;
- Uma tomada só tem utilidade quando algum aparelho é conectado a ela;
- Conectando o aparelho, é fechado um circuito por onde passa a eletricidade que chega até ele.

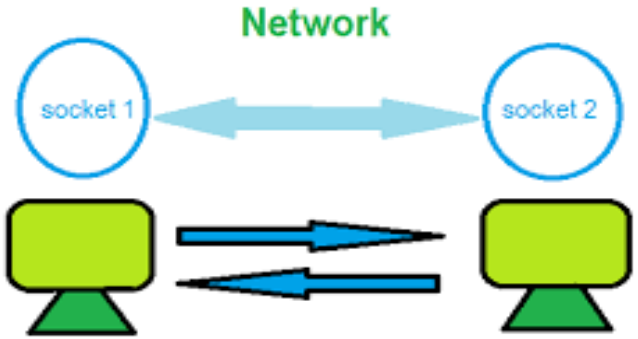
Plug & socket type N

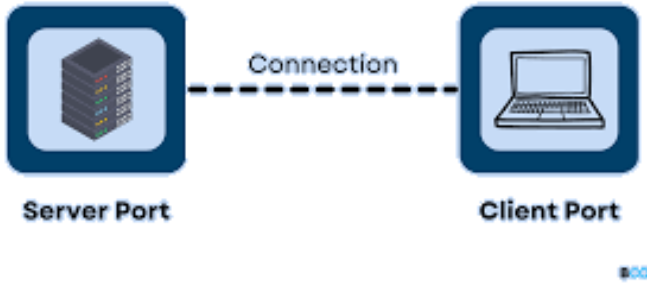
Plug type C

Sockets

Definição

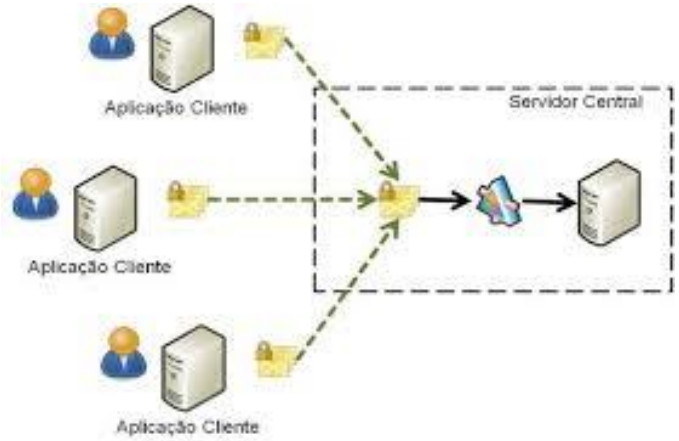
- Em programação, um *Socket* representa um canal de comunicação bidirecional entre duas aplicações, com um *socket* de um lado conectado ao *socket* do outro;
- Uma vez conectados, é possível trafegar dados entre esses dois *sockets* e, consequentemente, entre suas aplicações;
- Fica a critério das aplicações envolvidas enviarem e/ou receberem dados através dos seus *sockets* conectados.





Definição

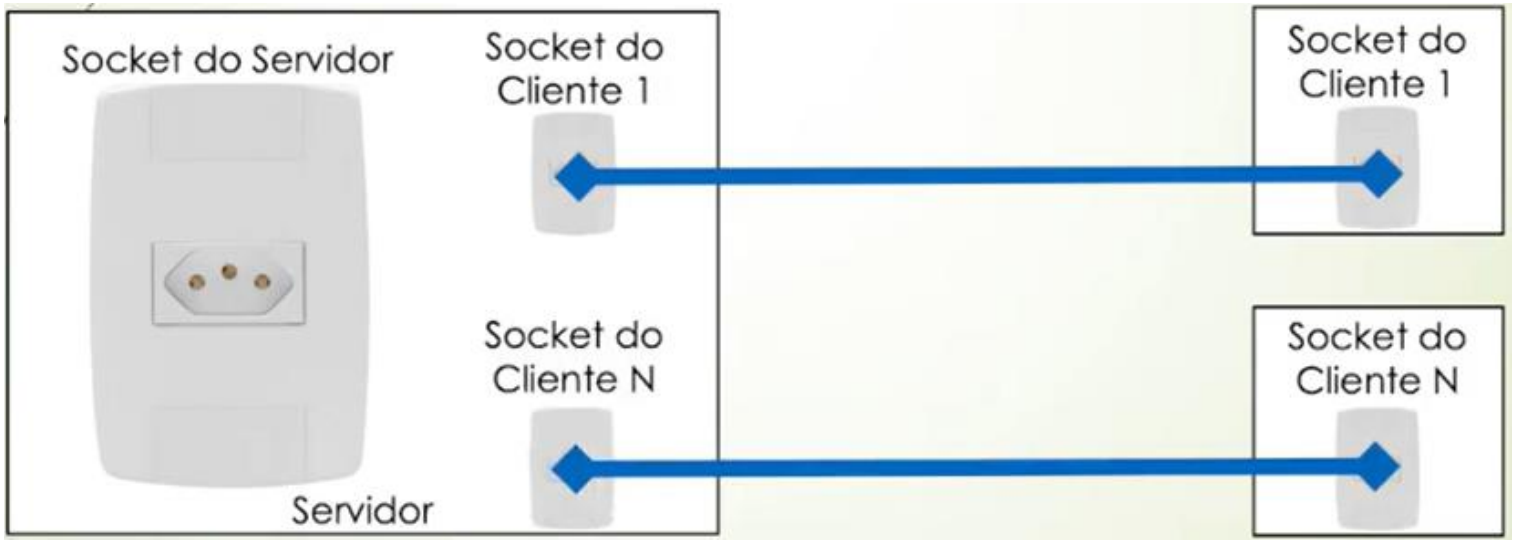
- Uma aplicação servidora terá um *socket* no qual as aplicações cliente se conectam a ela;
- Tendo um *socket* aberto no servidor, é aberta uma **porta** por onde as conexões com os clientes podem ser estabelecidas;
- Pode-se dizer que o servidor fica "escutando" essa tal porta;
- O cliente se conecta nesse *socket*, estabelecendo o canal de comunicação.



Sockets

Definição

- Cada cliente, ao se conectar a um determinado servidor, cria um *socket* para enviar e receber dados relacionados a esse servidor;
- Para cada cliente conectado, o servidor também cria um *socket* para receber e enviar dados relacionados a esse cliente;



Sockets

Definição

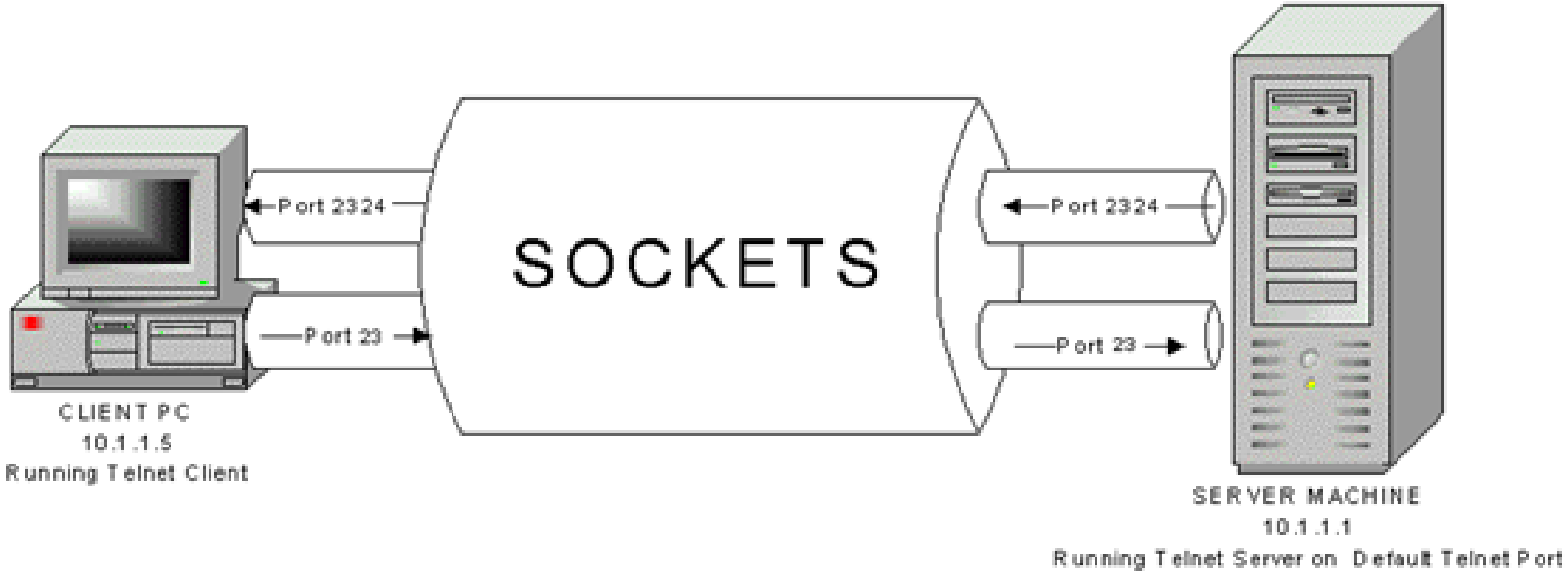
- Cada ***Socket*** precisa estar associado a um **Endereço *IP*** – *Internet Protocol* e a um número de **Porta**, para ser possível que outras aplicações possam se conectar a ele e trocar mensagens;
- Um **Porta** é um Ponto Final – ***Endpoint*** de Comunicação.

Definição

- Sem o número da **Porta**, o SO não tem como saber para qual aplicação deve entregar uma mensagem recebida;
- Na analogia adotada, sendo o Endereço IP o Endereço do Prédio e a Porta o Número do Apartamento, só com o Endereço *IP*, **sem a Porta**, é como ter o Endereço do Prédio mas não ter o Número do Apartamento, ou seja, **a Mensagem não será entregue**!

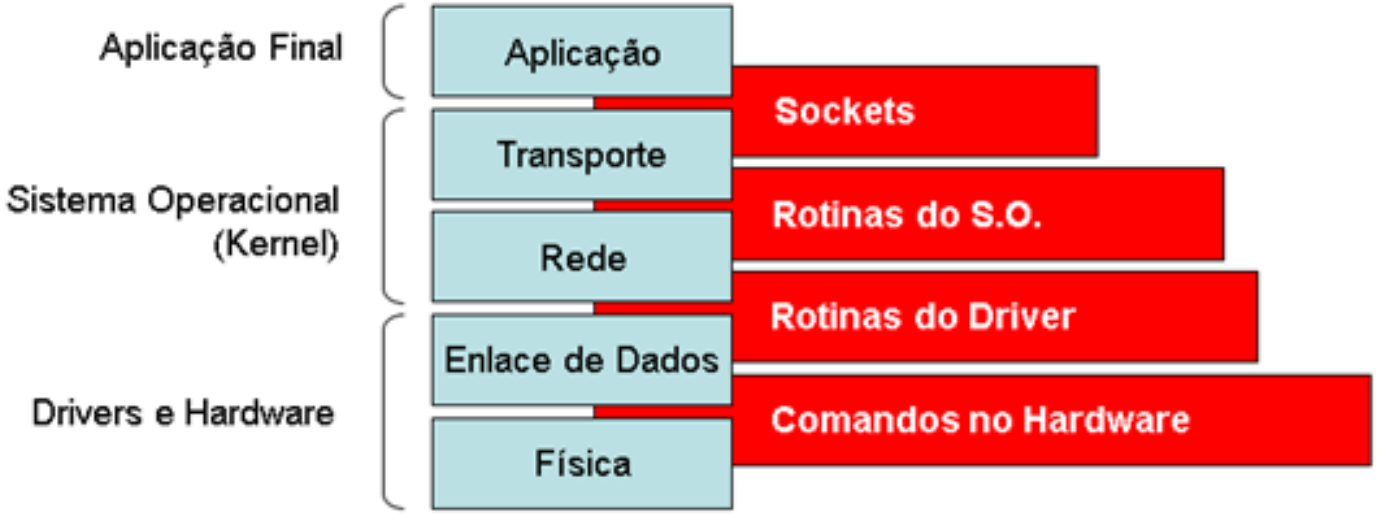
Definição

- Representação da comunicação via *socket* entre dois computadores distintos, cliente e servidor, bem como suas Portas e seus *IPs*:



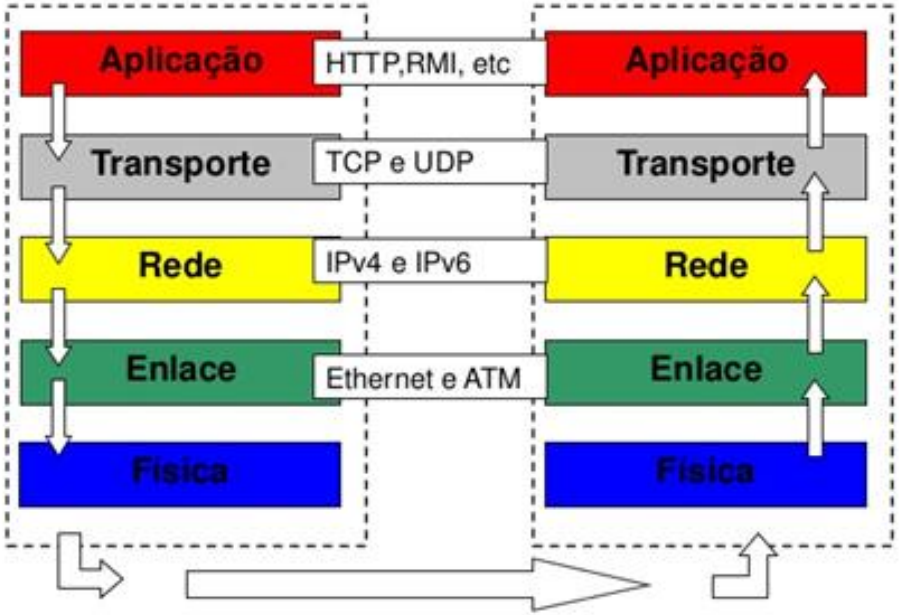
Definição

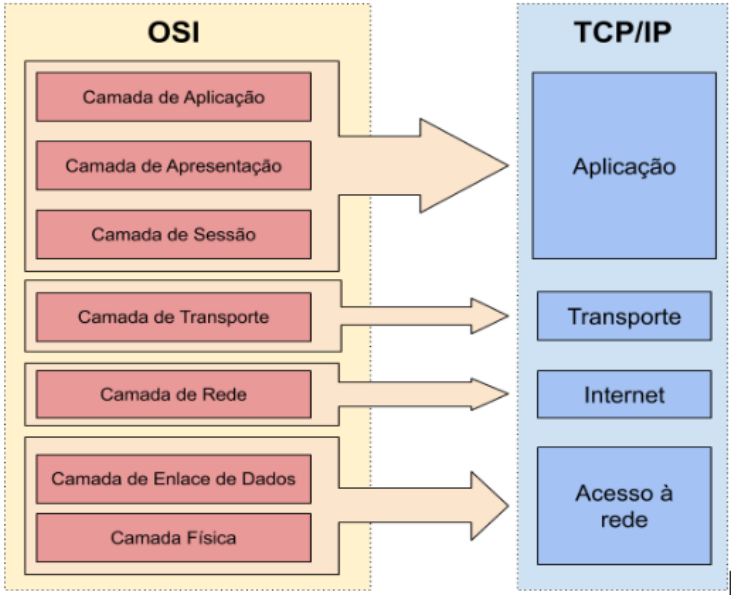
- Conjunto de etapas na qual a comunicação via *socket* passa dentro de um computador:



Definição

- Tipos de Serviços oferecidos pelas camadas de comunicação simplificadas e Modelo *OSI* x *TCP*/*IP*:





Cliente-Servidor com Java Sockets

Tópico

- *Stream*

Stream

Definição

- *Stream*, em Inglês, significa **corrente**, **fluxo** ou **riacho**;
- Um *Stream* é usado em programação para permitir imprimir, ler e gravar dados de/em dispositivos de entrada – *input* e saída – *output*;
- Ao usar um *Stream*, pode-se enviar ou receber os dados por meio de um fluxo;
- São exemplos de *Stream*, os arquivos e os canais de comunicação, como os *sockets*;
- No contexto apresentado, esses dispositivos serão os canais de comunicação entre o cliente e o servidor, ou seja, os *sockets*;

Stream

Definição

- A partir do *Java 8*, o conceito de *Stream* foi ampliado para se referir a um novo recurso da linguagem, utilizado para realizar o processamento dos dados de forma sequencial (serial) ou simultânea (paralela);
- Portanto, o conceito de *Stream*, do *Java 8* em diante, depende do contexto, podendo se referir ao fluxo de entrada e saída de dados, ou ao processamento de dados sequencial ou simultâneo. Atenção, então!

Cliente-Servidor com Java Sockets

Exemplo

- Projeto do Servidor *Java* – sem Interface gráfica:

```
1  import java.io.IOException;
2  import java.net.Socket;
3  import java.net.ServerSocket;
4  import java.util.Scanner;
5
6  public class SimpleServerTest
7  {  // Especifica endereco do IP local para o servidor
8        public static final String ENDERECO = "127.0.0.1";
9     // Especifica porta livre do computador para o servidor escutar
10       public static final int PORTA = 3334;
11
12    public static void   main(String args[])
13    {  try
14       {  // Cria servico de escuta da porta especificada via socket no servidor
15          ServerSocket   servidor = new ServerSocket(PORTA);
16          System.out.println("Servidor iniciado na porta " + PORTA);
17          // Cria canal de comunicacao para a conexao de um cliente (servico)
18          Socket   cliente = servidor.accept();
19          // Apresenta endereco do cliente conectado ao servidor
20          System.out.println("Cliente do IP: " +
21             cliente.getInetAddress().getHostAddress() +
22             " conectado ao servidor");
```

Cliente-Servidor com Java Sockets

Exemplo

- Projeto do Servidor *Java* – sem Interface gráfica (continuação):

```
23          // Obtem a entrada do canal (socket)
24          Scanner entrada = new Scanner(cliente.getInputStream());
25          // Escutando as mensagens do cliente que chegam ao servidor
26          System.out.println("Mensagens do Cliente:");
27          while(entrada.hasNextLine())  // Enquanto a entrada tiver uma próxima linha...
28          {  System.out.println(entrada.nextLine());   // Imprime na tela a mensagem de entrada
29          }
30          // Fechando o canal de entrada do servidor e o servico de conexao ao servidor
31          System.out.println("Servidor finalizado!");
32          entrada.close();  // Fecha o canal de comunicacao
33          servidor.close(); // Fecha o servico
34       }
35       catch(IOException ex)
36       {  System.out.println("Erro ao criar o servidor!");
37       }
38    }
39 }
40
```

Cliente-Servidor com Java Sockets

Exemplo

- Projeto do Cliente *Java* – Interface *Swing* simples:

```
1  import java.io.IOException;
2  import java.io.PrintStream;
3  import java.net.Socket;
4  import javax.swing.JOptionPane;
5
6  public class SimpleClientTest
7  {  // Criando a variavel de conexao do tipo Socket
8     private static Socket cliente;   // Comunicacao com o servidor
9     public static void main(String args[])
10    {  try
11       {  String msg;
12          iniciaCliente();  // cria cliente e inicia conexao com o servidor
13          System.out.println("Mensagens para o servidor:");
14          do
15          {  msg = JOptionPane.showInputDialog("Digite mensagem (ou <sair> para encerrar)");
16             if(!msg.equalsIgnoreCase("sair"))
17             {  System.out.println(msg);
18                enviaMensagem(msg);
19             }
20          }while(!msg.equalsIgnoreCase("sair"));
21          System.out.println("Cliente se desconectou do servidor!");
22       }
23       catch(IOException ex)
24       {  System.out.println("Falha na comunicacao: " + ex.getMessage());
25       }
26    }
27
```

Cliente-Servidor com Java Sockets

Exemplo

- Projeto do Cliente *Java* – Interface *Swing* simples (continuação):

```
28    private static void iniciaCliente() throws IOException
29    {  // Cria um cliente com IP e Porta para a comunicacao com o servidor
30       cliente = new Socket(SimpleServerTest.ENDERECO, SimpleServerTest.PORTA);
31       System.out.println("Cliente: " + SimpleServerTest.ENDERECO +
32          ":" + SimpleServerTest.PORTA + " conectado ao servidor!");
33    }
34
35    private static void enviaMensagem(String msg) throws IOException
36    {  // Objeto para enviar mensagem ao servidor
37       PrintStream saida = new PrintStream(cliente.getOutputStream());
38       // Envia mensagem ao servidor
39       saida.println(msg);
40    }
41 }
42
```

Características

- Os códigos apresentados são aplicações simples da arquitetura cliente-servidor em;
- O código do servidor *Java*não apresenta interface gráfica, apenas mensagens de *status* via console;
- O código do cliente apresenta interface com o usuário simples, do tipo *Swing*, além de apresentar mensagens de status via console, também;
- Essas aplicações não permitem a conexão de mais de um cliente por vez;
- Essas aplicações não permitem o envio de mensagens do servidor para o cliente, somente do cliente para o servidor.

Conclusões

- O uso de aplicações simples em Java para arquitetura cliente-servidor pode ser uma escolha viável, dependendo dos requisitos do projeto e das necessidades específicas;
- Aqui estão algumas conclusões e considerações a serem levadas em conta:
  1. Portabilidade: Uma das vantagens da linguagem de programação Java é sua portabilidade, o que significa que as aplicações Java podem ser executadas em diferentes sistemas operacionais, desde que haja uma máquina virtual Java (*JVM*) disponível e isso facilita a distribuição da aplicação em uma variedade de plataformas;

Conclusões

- Aqui estão algumas conclusões e considerações a serem levadas em conta:
  2. Desempenho: O desempenho de aplicações cliente-servidor na linguagem de programação Java pode ser adequado para muitos casos de uso, especialmente se você otimizar o código e o *design* da aplicação, no entanto, para cenários de alto desempenho, devem ser consideradas outras linguagens de programação mais adequadas, dependendo dos requisitos específicos;

Conclusões

- Aqui estão algumas conclusões e considerações a serem levadas em conta:
  3. Facilidade de desenvolvimento: A linguagem de programação *Java* e sua vasta biblioteca padrão tornam mais fácil o desenvolvimento de aplicações cliente-servidor, existindo muitos *frameworks* e bibliotecas disponíveis que podem simplificar tarefas comuns, como comunicação de rede e gerenciamento de conexões;

Conclusões

- Aqui estão algumas conclusões e considerações a serem levadas em conta:

4. Segurança: A linguagem de programação Java oferece recursos de segurança robustos, como o modelo de segurança de classe e a capacidade de executar código em um ambiente controlado, *i.e. sandbox*, que pode ser benéfico ao lidar com aplicações cliente-servidor que precisam garantir a integridade e a segurança dos dados transmitidos;

Conclusões

- Aqui estão algumas conclusões e considerações a serem levadas em conta:

5. Escalabilidade: A arquitetura cliente-servidor em Java pode ser dimensionada para atender a demandas crescentes, desde que o *design* da aplicação seja adequado para escalabilidade, podendo envolver o uso de servidores de aplicativos *Java EE*, balanceamento de carga e outras técnicas de escalabilidade;

Conclusões

- Aqui estão algumas conclusões e considerações a serem levadas em conta:

6. Conectividade de banco de dados: A linguagem de programação Java oferece suporte robusto para conexões a bancos de dados, permitindo que as aplicações cliente-servidor acessem e manipulem dados de forma eficiente;

Conclusões

- Aqui estão algumas conclusões e considerações a serem levadas em conta:

7. Manutenção e evolução: A linguagem de programação *Java* é uma linguagem madura e amplamente adotada, o que significa que você pode encontrar recursos e ferramentas para manter e evoluir suas aplicações ao longo do tempo.

Conclusões

- Aqui estão algumas conclusões e considerações a serem levadas em conta:

8. Restrições: É importante notar que a escolha de usar Java para aplicações cliente-servidor depende dos requisitos específicos do projeto. Outras linguagens de programação, como *Python*, *C#, JavaScript*, entre outras, também são populares para desenvolvimento cliente-servidor e podem ser mais adequadas dependendo das circunstâncias. Além disso, a arquitetura cliente-servidor em si deve ser projetada com cuidado para atender aos objetivos do sistema em termos de desempenho, escalabilidade, segurança e outros aspectos importantes.

Cliente-Servidor com Java Sockets

Exercício 1

a. Criar um projeto denominado ***SimpleClientServer*** na IDE de sua preferência e digitar as classes fornecidas ***SimpleServerTest.java*** e ***SimpleClientTest.java***;

b. Executar, somente, a classe ***SimpleClientTest.java***, verificar e registrar o que ocorre. Explique, em detalhes, o que ocorre;

c. Executar a classe ***SimpleServerTest.java***, verificar e registrar o que ocorre. Explique, em detalhes, o que ocorre;

d. Sem encerrar a execução da classe ***SimpleServerTest.java***, executar, a seguir, a classe ***SimpleClientTest.java***, digitando algumas mensagens na interface com o usuário, uma após a outra e, por fim, optar por sair. Verificar e registrar o que ocorre. Explique, em detalhes, o que ocorre;

Exercício 1

e. Sem encerrar a execução da classe ***SimpleServerTest.java***, executar, a seguir, a classe ***SimpleClientTest.java***, digitando uma mensagem, apenas, na interface com o usuário e não optar por sair. Sem encerrar a execução da classe ***SimpleServerTest.java***, nem da primeira instância de execução da classe ***SimpleClientTest.java***, executar outra instância da classe ***SimpleClientTest.java***. Digitar algumas mensagens na segunda instância da classe ***SimpleClientTest.java***, sem sair dela, verificar e registrar o que ocorre. Repetir a digitação para a primeira instância da classe ***SimpleClientTest.java***, sem sair dela também. Explique, em detalhes, o que ocorre;

Exercício 1

f. Encerrar a execução da segunda instância da classe ***SimpleClientTest.java***, digitando algumas mensagens na primeira instância da classe ***SimpleClientTest.java***. Verificar e registrar o que ocorre. Explique, em detalhes, o que ocorre;

g. Executar uma segunda instância da classe ***SimpleServerTest.java***, ainda com a primeira instância dessa classe em execução. Verificar e registrar o que ocorre. Explique, em detalhes, o que ocorre.

Exercício 2

- Criar uma interface gráfica para a classe ***SimpleClientTest.java***, através dos modelos de *layout* estudados anteriormente nesta disciplina (***flowlayout***, ***borderlayout***, ***gridlayout*** ou os automaticamente gerados pela IDE ***NetBeans***), eliminando toda e qualquer forma de comunicação com o usuário através do **console** e/ou por interface ***Swing*** como foi fornecida, criando nessa nova interface gráfica os campos específicos para as mensagens a serem enviadas ao servidor e para as mensagens de *status* da comunicação, além da inclusão dos botões e suas funcionalidades para **Enviar**, **Limpar** e **Sair**, na própria interface.

Exercício Desafio

- Estudar, pesquisar, desenvolver e testar novas classes ***SimpleServer2Test.java***, e ***SimpleClient2Test.java***, baseadas nas respectivas classes fornecidas, fazendo com que a nova classe servidor retransmita de volta para a nova classe cliente, as mensagens que a classe servidor receber, vindas da classe cliente, apresentando-as, também, na interface gráfica da classe cliente desenvolvida no exercício 2 desta atividade, junto com as suas mensagens anteriores.

Bibliografia Básica

- MILETTO, Evandro M.; BERTAGNOLLI, Silvia de Castro. Desenvolvimento de software II: introdução ao desenvolvimento web com HTML, CSS, javascript e PHP (Tekne). Porto Alegre: Bookman, 2014. E-book. Referência Minha Biblioteca: https://integrada.minhabiblioteca.com.br/#/books/9788582601969
- WINDER, Russel; GRAHAM, Roberts. Desenvolvendo Software em Java, 3ª edição. Rio de Janeiro: LTC, 2009. E-book. Referência Minha Biblioteca: https://integrada.minhabiblioteca.com.br/#/books/978-85-216-1994-9
- DEITEL, Paul; DEITEL, Harvey. Java: how to program early objects. Hoboken, N. J: Pearson, c2018. 1234 p. ISBN 9780134743356.

Continua...

Bibliografia Básica (*continuação*)

- HORSTMANN, Cay S; CORNELL, Gary. Core Java. SCHAFRANSKI, Carlos (Trad.), FURMANKIEWICZ, Edson (Trad.). 8. ed. São Paulo: Pearson, 2010. v. 1. 383 p. ISBN 9788576053576.
- LIANG, Y. Daniel. Introduction to Java: programming and data structures comprehensive version. 11. ed. New York: Pearson, c2015. 1210 p. ISBN 9780134670942.
- TURINI, Rodrigo. Desbravando Java e orientação a objetos: um guia para o iniciante da linguagem. São Paulo: Casa do Código, [2017]. 222 p. (Caelum).

Bibliografia Complementar

- HORSTMANN, Cay. Conceitos de Computação com Java. Porto Alegre: Bookman, 2009. E-book. Referência Minha Biblioteca: https://integrada.minhabiblioteca.com.br/#/books/9788577804078
- MACHADO, Rodrigo P.; FRANCO, Márcia H. I.; BERTAGNOLLI, Silvia de Castro. Desenvolvimento de software III: programação de sistemas web orientada a objetos em java (Tekne). Porto Alegre: Bookman, 2016. E-book. Referência Minha Biblioteca: https://integrada.minhabiblioteca.com.br/#/books/9788582603710
- BARRY, Paul. Use a cabeça! Python. Rio de Janeiro: Alta Books, 2012. 458 p.
  ISBN 9788576087434.

Continua...

Bibliografia Complementar (continuação)

- LECHETA, Ricardo R. Web Services RESTful: aprenda a criar Web Services RESTfulem Java na nuvem do Google. São Paulo: Novatec, c2015. 431 p.
  ISBN 9788575224540.
- SILVA, Maurício Samy. JQuery: a biblioteca do programador. 3. ed. rev. e ampl. São Paulo: Novatec, 2014. 544 p.
  ISBN 9788575223871.
- SUMMERFIELD, Mark. Programação em Python 3: uma introdução completa à linguagem Phython. Rio de Janeiro: Alta Books, 2012. 506 p.
  ISBN 9788576083849.

Continua...

Bibliografia Complementar (continuação)

- YING, Bai. Practical database programming with Java. New Jersey: John Wiley & Sons, c2011. 918 p.
- ZAKAS, Nicholas C. The principles of object-oriented JavaScript. San Francisco, CA: No Starch Press, c2014. 97 p. ISBN 9781593275402.
- CALVETTI, Robson. Programação Orientada a Objetos com Java. Material de aula, São Paulo, 2020.

Aula 20 – L1/1, L2/1 e L3/1

FIM

Engenharia da Computação – 3ª série

Cliente-Servidor com Java Sockets
(L1/1, L2/1 e L3/1)

2025

Aula 20 – L1/2, L2/2 e L3/2

Horário

Terça-feira: 2 x 2 aulas/semana

- L1/1 (07h40min-09h20min): *Prof. Calvetti*;
- L1/2 (09h30min-11h10min): *Prof. Calvetti*;
- L2/1 (07h40min-09h20min): *Prof. Menezes*;
- L2/2 (11h20min-13h00min): *Prof. Calvetti*;
- L3/1 (09h30min-11h10min): *Prof. Evandro*;
- L3/2 (11h20min-13h00min): *Prof. Evandro.*

Exercícios

- Terminar, entregar e apresentar ao professor para avaliação, os exercícios propostos na aula de teoria, deste material.

Bibliografia (apoio)

- LOPES, ANITA. GARCIA, GUTO. Introdução à Programação: 500 algoritmos resolvidos. Rio de Janeiro: Elsevier, 2002.
- DEITEL, P. DEITEL, H. Java: como programar. 8 Ed. São Paulo: Prentice-Hall (Pearson), 2010;
- BARNES, David J.; KÖLLING, Michael. Programação orientada a objetos com Java: uma introdução prática usando o BlueJ . 4. ed. São Paulo: Pearson Prentice Hall, 2009.

Aula 20 – L1/2, L2/2 e L3/2

FIM