ECM251 – Linguagens de Programação I

Aula 23 – L1/1 e L2/1

Engenharia da Computação – 3ª série

Cliente-Servidor Java Sockets Bate-Papo (L1/1 – L2/1)

2023

Aula 23 – L1/1 e L2/1

Horário

Terça-feira: 2 aulas/semana

- L1/1 (07h40min-09h20min): *Prof. Calvetti*;
- L2/1 (07h40min-09h20min): *Prof. Igor Silveira*;

Cliente-Servidor Java Sockets Bate-Papo

Tópico

- Aplicativo de Bate-Papo

Aplicativo de Bate-Papo

Definição

- Uma das soluções de arquitetura **Cliente-Servidor** apresentadas anteriormente, por motivos didáticos, implementava, inicialmente, um **Servidor** cuidando da conexão com um único **Cliente**, através de em uma única ***Thread***;
- Posteriormente, incrementando-se a complexidade do código, para que um **Servidor** pudesse cuidar de múltiplas conexões, para vários **Clientes**, passou-se a utilizar na implementação, várias ***Threads*** concorrentes, uma por conexão e **Cliente**;
- Para se elaborar um aplicativo de **Bate-Papo**, utilizando a mesma arquitetura **Cliente-Servidor**, ainda com ***Sockets Java***, é necessária, então, a implementação da **comunicação bidirecional** entre o **Servidor** e cada um dos seus **Clientes**.

Aplicativo de Bate-Papo

Definição

- Com visto, existem, em **Java**, vários métodos utilizados de forma **Bloqueante**, que aguardam a ocorrência de um determinado evento, "travando" o fluxo de execução de sua ***Thread***, como, por exemplo, os métodos ***nextLine()***, ***getMessage()*** etc.;
- Para a comunicação entre **Cliente-Servidor**, com a possibilidade de conexão de vários **Clientes** concorrentes ao **Servidor**, os métodos **bloqueantes** são utilizados em conjunto com o conceito de ***Multithreading***.

Aplicativo de Bate-Papo

Exemplo

- Métodos **Bloqueantes** utilizados pelo **Servidor** e pelo **Cliente** anteriormente:

```
public void clientMessageLoop(ClientSocket clientSocket)
{  String msg;
   try
   {  while((msg = clientSocket.getMessage()) != null && !msg.equalsIgnoreCase("sair"))
      {  System.out.printf("Mensagem recebida do cliente %s: %s\n", clientSocket.getRemoteSocketAddress(), msg);
      }
   }
   finally
   {  clientSocket.close();
   }
}
```

```
private void messageLoop() throws IOException
{  String msg;
   System.out.println("Aguardando a digitação de uma mensagem!");
   do
   {  System.out.print("Digite uma mensagem (ou <sair> para finalizar): ");
      msg = scanner.nextLine();
      saida.println(msg);
   }while(!msg.equalsIgnoreCase("sair"));
}
```

Aplicativo de Bate-Papo

Exemplo

- ***Multithreading*** utilizado pelo **Servidor** anteriormente:

```
private void clientConnectionLoop() throws IOException
{  System.out.println("Aguardando conexao de um cliente!");
   do
   {  ClientSocket clientSocket = new ClientSocket(serverSocket.accept());
      new Thread(() -> clientMessageLoop(clientSocket)).start();  // Criando uma Expressão Lambda
   }while(true);
}

public void clientMessageLoop(ClientSocket clientSocket)
{  String msg;
   try
   {  while((msg = clientSocket.getMessage()) != null && !msg.equalsIgnoreCase("sair"))
      {  System.out.printf("Mensagem recebida do cliente %s: %s\n", clientSocket.getRemoteSocketAddress(), msg);
      }
   }
   finally
   {  clientSocket.close();
   }
}
```

Conclusão

- Seguindo essa linha de solução, no caso de haver a necessidade de mais de um **Cliente** tentando se conectar ao mesmo **Servidor** e da implementação desse **Servidor** estar utilizando uma API de ***Sockets*** com operação **bloqueante**, como foi o caso, uma das soluções possíveis é que o **Servidor** seja implementado utilizando ***Threads*** separadas, cada uma cuidando da comunicação com um **Cliente** distinto;
- Daí, para a implementação de um aplicativo de **Bate-Papo**, é necessário, além da comunicação **bidirecional** e do uso de métodos **Bloqueantes**, a implementação de ***Multithreads*** do lado do **Servidor** e, também, do lado do **Cliente**.

Cliente-Servidor Java Sockets Bate-Papo

Tópico

- Cliente-Servidor Java Sockets Bate-Papo

Características

- Os códigos apresentados são aplicações simples da arquitetura **Cliente-Servidor** em **Java**;
- O código do **Servidor** não apresenta interface gráfica, apenas mensagens de ***status***, via console;
- O código do **Cliente** apresenta interface com o usuário simples, através do uso de mensagens digitadas e recebidas via console, também;
- Essas aplicações permitem o envio de mensagens do **Servidor** para os **Clientes**, além do envio dos **Clientes** para o **Servidor**;
- A aplicação de **Bate-Papo** permite conexões de mais de um **Cliente** por vez ao **Servidor**.

Exemplo

- Projeto Java Multicliente-Servidor com *Socket* e *Threads*, em ambos os lados, com comunicação Bidirecional entre eles:

Classe *SocketCliente.java*

Cliente-Servidor Java Sockets Bate-Papo

Exemplo

- Projeto Java Multicliente-Servidor com *Socket* e *Threads*, em ambos os lados, com comunicação Bidirecional entre eles:

```
1  import java.io.*;
2  import java.net.Socket;
3  import java.net.SocketAddress;
4
5  public class SocketCliente
6  {   private final Socket socket;
7      private final BufferedReader entrada;
8      private final PrintWriter saida;
9
10     public SocketCliente(final Socket socket) throws IOException
11     {   this.socket = socket;
12         System.out.println("Conectado com " + socket.getRemoteSocketAddress() + "!");
13         this.entrada = new BufferedReader(new InputStreamReader(socket.getInputStream()));
14         this.saida = new PrintWriter(socket.getOutputStream(), true);
15     }
16
17     public SocketAddress getRemoteSocketAddress()
18     {   return socket.getRemoteSocketAddress();
19     }
20
```

Cliente-Servidor Java Sockets Bate-Papo

Exemplo

- Projeto Java Multicliente-Servidor com *Socket* e *Threads*, em ambos os lados, com comunicação Bidirecional entre eles:

```
21      public void close()
22      {   try
23          {   entrada.close();
24              saida.close();
25              socket.close();
26          }
27          catch(IOException ex)
28          {   System.out.println("Erro o fechar o socket: " + ex.getMessage());
29          }
30      }
31
32      public String getMessage()
33      {   try
34          {   return entrada.readLine();
35          }
36          catch(IOException ex)
37          {   return null;
38          }
39      }
40
41      public boolean sendMsg(String msg)
42      {   saida.println(msg);
43          return !saida.checkError();
44      }
45  }
```

Cliente-Servidor Java Sockets Bate-Papo

Exemplo

- Projeto Java Multicliente-Servidor com *Socket* e *Threads*, em ambos os lados, com comunicação Bidirecional entre eles:

Classe *ServidorBatepapo.java*

Cliente-Servidor Java Sockets Bate-Papo

Exemplo

- Projeto Java Multicliente-Servidor com *Socket* e *Threads*, em ambos os lados, com comunicação Bidirecional entre eles:

```
1  import java.io.IOException;
2  import java.net.ServerSocket;
3  import java.util.List;
4  import java.util.LinkedList;
5  import java.util.Iterator;
6
7  public class ServidorBatepapo
8  {  public static final String ADDRESS = "127.0.0.1";  // IP Address local do servidor
9     public static final int PORT = 4000; //ou 3334
10    private ServerSocket serverSocket;
11    private final List<SocketCliente> clients = new LinkedList<>();
12
13    public void start() throws IOException
14    {  serverSocket = new ServerSocket(PORT);
15       System.out.println("Servidor iniciado na porta: " + PORT);
16       clientConnectionLoop();
17    }
18
```

Cliente-Servidor Java Sockets Bate-Papo

Exemplo

- Projeto Java Multicliente-Servidor com *Socket* e *Threads*, em ambos os lados, com comunicação Bidirecional entre eles:

```
19    private void clientConnectionLoop() throws IOException
20    {  System.out.println("Aguardando conexao de um cliente!");
21       while(true)
22       {  SocketCliente clientSocket = new SocketCliente(serverSocket.accept());
23          clients.add(clientSocket);
24          new Thread(() -> clientMessageLoop(clientSocket)).start();  // Expressão Lambda
25       }
26    }
27
28    private void clientMessageLoop(SocketCliente clientSocket)
29    {  String msg;
30       try
31       {  while((msg = clientSocket.getMessage()) != null)
32          {  if("sair".equalsIgnoreCase(msg)) return;
33             System.out.printf("<- Cliente %s: %s\n", clientSocket.getRemoteSocketAddress(), msg);
34             sendMsgToAll(clientSocket, msg);
35          }
36       }
37       finally
38       {  clientSocket.close();
39       }
40    }
41
```

Cliente-Servidor Java Sockets Bate-Papo

Exemplo

- Projeto Java Multicliente-Servidor com *Socket* e *Threads*, em ambos os lados, com comunicação Bidirecional entre eles:

```
42    private void sendMsgToAll(SocketCliente sender, String msg)
43    {  Iterator<SocketCliente> iterator = clients.iterator();
44       while(iterator.hasNext())
45       {  SocketCliente clientSocket = iterator.next();
46          if(!sender.equals(clientSocket))
47          {  if(!clientSocket.sendMsg("Cliente " + sender.getRemoteSocketAddress() + ": " + msg))
48             {  iterator.remove();
49             }
50          }
51       }
52    }
53    public static void main(String args[])
54    {  System.out.println("*v*v*v* CONSOLE DO SERVIDOR *v*v*v*");
55       try
56       {  ServidorBatepapo server = new ServidorBatepapo();
57          server.start();
58       }
59       catch(IOException ex)
60       {  System.out.println("Erro ao iniciar o servidor: " + ex.getMessage());
61       }
62       System.out.println("Servidor finalizado!");
63    }
64 }
```

Cliente-Servidor Java Sockets Bate-Papo

Exemplo

- Projeto Java Multicliente-Servidor com *Socket* e *Threads*, em ambos os lados, com comunicação Bidirecional entre eles:

Classe *ClienteBatepapo.java*

Cliente-Servidor Java Sockets Bate-Papo

Exemplo

- Projeto Java Multicliente-Servidor com *Socket* e *Threads*, em ambos os lados, com comunicação Bidirecional entre eles:

```
1  import java.io.IOException;
2  import java.net.Socket;
3  import java.util.Scanner;
4
5  public class ClienteBatepapo implements Runnable
6  {  private SocketCliente clientSocket;
7     private Scanner scanner;
8
9     public ClienteBatepapo()
10    {  scanner = new Scanner(System.in);
11    }
12
13    public void start() throws IOException
14    {  try
15       {  clientSocket = new SocketCliente(new Socket(ServidorBatepapo.ADDRESS, ServidorBatepapo.PORT));
16          new Thread(this).start();
17          messageLoop();
18       }
19       finally
20       {  clientSocket.close();
21       }
22    }
23
```

Cliente-Servidor Java Sockets Bate-Papo

Exemplo

- Projeto Java Multicliente-Servidor com *Socket* e *Threads*, em ambos os lados, com comunicação Bidirecional entre eles:

```
24    @Override
25    public void run()
26    {  String msg;
27       while((msg = clientSocket.getMessage()) != null)
28       {  System.out.printf("\n-> %s\n", msg);
29          System.out.print("Digite uma mensagem (ou <sair> para finalizar):\n<- ");
30       }
31    }
32
33    private void messageLoop() throws IOException
34    {  String msg;
35       System.out.println("Digite uma mensagem (ou <sair> para finalizar):");
36       do
37       {  System.out.print("<- ");
38          msg = scanner.nextLine();
39          clientSocket.sendMsg(msg);
40       }while(!msg.equalsIgnoreCase("sair"));
41    }
42
```

Cliente-Servidor Java Sockets Bate-Papo

Exemplo

- Projeto Java Multicliente-Servidor com *Socket* e *Threads*, em ambos os lados, com comunicação Bidirecional entre eles:

```
43    public static void main(String args[])
44    {  System.out.println("*v*v*v* CONSOLE DO CLIENTE *v*v*v*");
45       try
46       {  ClienteBatepapo client = new ClienteBatepapo();
47          client.start();
48       }
49       catch(IOException ex)
50       {  System.out.println("Erro ao iniciar o cliente: " + ex.getMessage());
51       }
52       System.out.println("Cliente finalizado!");
53    }
54 }
55
```

Conclusões

- Usar ***Threads*** no servidor permite que ele atenda a vários clientes simultânea e independentemente;
- O uso de aplicações em **Java,** com **Múltiplos Clientes** simultâneos conectados a um servidor, pode ser uma escolha viável para arquitetura **Cliente-Servidor**, dependendo dos requisitos do projeto e das necessidades específicas;
- A possibilidade de vários **Clientes** se conectarem a um único **Servidor** ao mesmo tempo e se comunicarem de forma **Bidirecional**, abre um número grande de aplicações, uso e possibilidades de comunicação entre **Cliente-Servidor**.

Conclusões

- Essa solução com um **Servidor** utilizando múltiplas ***Threads***, uma para cada **Cliente** conectado a ele, comunicando-se de forma **Bidirecional**, é parcialmente escalável, pois sempre haverá um limite de ***Threads*** permitidas pelo Sistema Operacional – SO e, consequentemente, um limite de **Clientes** conectados a esse **Servidor** simultaneamente, sendo, mesmo assim, bastante adequada para as aplicações onde sabe-se que não será atingido esse limite de conexões.

Exercícios

1. No tópico **Cliente-Servidor Java Sockets Bate-Papo**, criar um projeto denominado ***ProjetoBatepapo***, na IDE de sua preferência, digitar as classes fornecidas ***SocketClient.java***, ***ServidorBatepapo.java*** e ***ClienteBatepapo.java***, distribuídas entre as páginas 11 à 21 deste material, executar, analisar, concluir e registrar o funcionamento das mesmas;
2. O que acontece se na aplicação do Exercício 1, o número de **Clientes** instanciados for elevado? Há um limite para a quantidade de **Clientes** na aplicação do **Servidor** fornecida? Justifique todas as respostas!

Exercícios

3. Com base no ***ProjetoBatepapo*** do Exercício 1 deste material, executar, somente, a classe ***ClienteBatePapo.java***, verificar, registrar e explicar, em detalhes, o ocorrido;
4. Com base no ***ProjetoBatepapo*** do Exercício 1 deste material, executar a classe ***ServidorBatepapo.java***, verificar, registrar e explicar, em detalhes, a operação do **Servidor**, sem sair dele;
5. Com base no ***ProjetoBatepapo*** do Exercício 1 deste material, com o **Servidor** em operação, executar uma primeira instância da classe ***ClienteBatepapo.java***, digitando algumas mensagens no seu console e, por fim, sair do **Cliente**, sem sair do **Servidor**. Verificar, registrar e explicar, em detalhes, as operações do **Cliente** e do **Servidor**;

Exercícios

6. Com base no ***ProjetoBatepapo*** do Exercício 1 deste material, com o **Servidor** em operação, executar, novamente, uma primeira instância da classe ***ClienteBatepapo.java***, digitando algumas mensagens no seu console e, por fim, não sair do **Cliente**, nem do **Servidor**. Verificar, registrar e explicar, em detalhes, as operações do **Cliente** e do **Servidor**;
7. Com base no ***ProjetoBatepapo*** do Exercício 1 deste material, com o **Servidor** e a primeira instância do **Cliente** em operação, executar uma segunda instância da classe ***ClienteBatepapo.java***, digitando algumas mensagens no seu console e no console da instância do outro **Cliente**, alternadamente e, por fim, não sair dos **Clientes**, nem do **Servidor**. Verificar, registrar e explicar, em detalhes, as operações dos **Clientes** e do **Servidor**;

Exercícios

8. Com base no ***ProjetoBatepapo*** do Exercício 1 deste material, com o **Servidor** e as duas instâncias dos **Clientes** em operação, executar uma terceira instância da classe ***ClienteBatepapo.java***, digitando algumas mensagens no seu console e nos consoles das instâncias dos outros **Clientes**, alternadamente e, por fim, não sair dos **Clientes**, nem do **Servidor**. Verificar, registrar e explicar, em detalhes, as operações dos **Clientes** e do **Servidor**;
9. Com base no ***ProjetoBatepapo*** do Exercício 1 deste material, com o **Servidor** e as três instâncias dos **Clientes** em operação, encerrar a execução das instâncias dos **Clientes**, uma por vez, digitando <***sair***> no console para cada uma delas, na ordem **Cliente** 1, 2 e 3 das instâncias. Verificar, registrar e explicar, em detalhes, as operações dos **Clientes** e do **Servidor**;

Exercícios

10. Com base no ***ProjetoBatepapo*** do Exercício 1 deste material, ainda com a primeira instância do **Servidor** em operação, executar, a seguir, uma segunda instância da classe ***Server.java***. Verificar, registrar e explicar, em detalhes, as operações dos **Servidores;**

Exercícios

11. Criar uma interface gráfica para a classe ***ClienteBatepapo.java***, através dos modelos de *layouts* 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***, criando nessa nova interface gráfica os campos específicos para as mensagens do **Cliente** que serão 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.

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.
- TANENBAUM, Andrew S.; MAARTEN, V. S. Sistemas Distribuídos: princípios e paradigmas, Pearson Education. 2ª edição. 2008.
- GOETZ et. al. Java Concurrency in Practice, 1st edition, 2006.
- CALVETTI, Robson. Programação Orientada a Objetos com Java. Material de aula, São Paulo, 2020.

Aula 23 – L1/1 e L2/1

FIM

Engenharia da Computação – 3ª série

Cliente-Servidor Java Sockets Bate-Papo (L1/2 – L2/2)

2023

Horário

Terça-feira: 2 aulas/semana

- L1/2 (09h30min-11h10min): *Prof. Calvetti*;
- L2/2 (11h20min-13h00min): *Prof. Calvetti*;

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 23 – L1/2 e L2/2

FIM