



Servidores e clientes baseados em Socket, com uso de Threads tanto no lado cliente quanto no lado servidor, acessando o banco de dados via JPA.

Leonardo Schaffer Mota – 2022.05.09098-1

Polo centro – Santo André – SP

Por que não paralelizar – 2023.2 – 2023.4

Objetivo da Prática

Criar servidores Java com base em Sockets.

Criar clientes síncronos para servidores com base em Sockets.

Criar clientes assíncronos para servidores com base em Sockets.

Utilizar Threads para implementação de processos paralelos.

1º Procedimento – Criando o Servidor e Cliente de Teste

CadastroServer:

```
5 package cadastroserver;
6
7 import controller.ProdutosJpaController;
8 import controller.UsuariosJpaController;
9 import java.io.IOException;
10 import java.net.ServerSocket;
11 import java.net.Socket;
12 import javax.persistence.EntityManagerFactory;
13 import javax.persistence.Persistence;
14
15 /**
16  * @author leosc
17  */
18 public class CadastroServer {
19
20     /**
21      * @param args the command line arguments
22      * @throws java.io.IOException
23      */
24     public static void main(String[] args) throws IOException {
25         EntityManagerFactory emf = Persistence.createEntityManagerFactory("CadastroServerPU");
26         ProdutosJpaController ctrl = new ProdutosJpaController(emf);
27         UsuariosJpaController ctrlUsu = new UsuariosJpaController(emf);
28
29         try (ServerSocket serverSocket = new ServerSocket(port: 4321)) {
30             System.out.println(x: "Servidor aguardando conexoes na porta 4321...");
31
32             while (true) {
33                 Socket socket = serverSocket.accept();
34                 CadastroThread thread = new CadastroThread(ctrl, ctrlUsu, s1: socket);
35                 thread.start(); // Inicia a thread
36                 System.out.println(x: "thread iniciado!");
37             }
38         }
39     }
40 }
41 }
```

CadastroThread:

```
5 package cadastroserver;
6
7 import controller.ProdutosJpaController;
8 import controller.UsuariosJpaController;
9 import java.io.IOException;
10 import java.io.ObjectInputStream;
11 import java.io.ObjectOutputStream;
12 import java.net.Socket;
13 import java.util.List;
14 import java.util.logging.Level;
15 import java.util.logging.Logger;
16 import model.Produtos;
17 import model.Usuarios;
18
19 /**
20  * @author leosc
21  */
22 public class CadastroThread extends Thread {
23     private final ProdutosJpaController ctrl;
24     private final UsuariosJpaController ctrlUsu;
25     private final Socket s1;
26
27     public CadastroThread(ProdutosJpaController ctrl, UsuariosJpaController ctrlUsu, Socket s1) {
28         this.ctrl = ctrl;
29         this.ctrlUsu = ctrlUsu;
30         this.s1 = s1;
31     }
32
33     @Override
34     public void run() {
35         try (ObjectOutputStream out = new ObjectOutputStream(out: s1.getOutputStream());
36             ObjectInputStream in = new ObjectInputStream(in: s1.getInputStream())) {
37
38             String login = (String) in.readObject();
39             String senha = (String) in.readObject();
40             List<Usuarios> usuariosList = ctrlUsu.findUsuariosEntities();
41             Usuarios usuarioAutenticado = null;
42
43             for (Usuarios usuario : usuariosList) {
44                 if (usuario.getLogin().equals(login) && usuario.getSenha().equals(senha)) {
45                     usuarioAutenticado = usuario;
46                     break;
47                 }
48             }
49
50             if (usuarioAutenticado == null) {
51                 System.out.println(x: "Credenciais inválidas. Desconectando cliente.");
52                 return;
53             }
54
55             System.out.println("Usuario autenticado: " + usuarioAutenticado.getLogin());
56
57             while (true) {
58                 String comando = (String) in.readObject();
59
60                 if (comando.equals("L")) {
61                     List<Produtos> produtos = ctrl.findProdutosEntities();
62                     out.writeObject(obj: produtos);
63                     System.out.println(x: "Enviando lista de produtos para o cliente.");
64                     break;
65                 }
66             }
67
68             try {
69                 if (out != null) {
70                     out.close();
71                 }
72                 if (in != null) {
73                     in.close();
74                 }
75                 if (s1 != null && !s1.isClosed()) {
76                     s1.close();
77                 }
78             } catch (IOException ex) {
79                 System.err.println("Erro ao fechar os fluxos e o socket: " + ex.getMessage());
80             }
81
82             } catch (IOException ex) {
83                 System.err.println("Erro de comunicação: " + ex.getMessage());
84             } catch (ClassNotFoundException ex) {
85                 Logger.getLogger(name: CadastroThread.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
86             }
87         }
88     }
89 }
```

CadastroClient:

```
5 package cadastroclient;
6
7 import java.io.IOException;
8 import java.io.ObjectInputStream;
9 import java.io.ObjectOutputStream;
10 import java.net.Socket;
11 import java.util.List;
12 import model.Produtos;
13
14 /**
15  *
16  * @author leosc
17  */
18 public class CadastroClient {
19
20     /**
21      * @param args the command line arguments
22      * @throws java.io.IOException
23      * @throws java.lang.ClassNotFoundException
24      */
25     public static void main(String[] args) throws IOException, ClassNotFoundException {
26         try {
27             Socket socket = new Socket("localhost", port: 4321);
28             ObjectOutputStream out = new ObjectOutputStream(out: socket.getOutputStream());
29             ObjectInputStream in = new ObjectInputStream(in: socket.getInputStream()); {
30
31                 out.writeObject(obj: "opl");
32                 out.writeObject(obj: "opl");
33                 out.writeObject(obj: "L");
34                 System.out.println(x: "Usuario conectado com sucesso");
35
36                 List<Produtos> produtos = (List<Produtos>) in.readObject();
37                 for (Produtos produto : produtos) {
38                     System.out.println(x: produto.getNome());
39                 }
40             }
41         }
42     }
43 }
```

Resultado da execução:

```
CadastroServer (run) X CadastroClient (run) X
run:
Servidor aguardando conexoes na porta 4321...
thread iniciado!
[EL Info]: 2023-08-25 18:00:39.69--ServerSession(1517433692)--EclipseL
Usuario autenticado: opl
Enviando lista de produtos para o cliente.
```

```
CadastroServer (run) X CadastroClient (run) X
run:
Usuario conectado com sucesso
Banana
Laranja
Manga
Batata
BUILD SUCCESSFUL (total time: 1 second)
```

a) Como funcionam as classes Socket e ServerSocket?

ServerSocket espera por conexões de clientes e, quando uma conexão é estabelecida, cria um Socket para a comunicação com esse cliente. Ambas as classes são fundamentais para a implementação de comunicação cliente-servidor em Java.

b) Qual a importância das portas para a conexão com servidores?

As portas são números de identificação associados aos processos de comunicação em um servidor. Elas são essenciais para direcionar dados para os serviços corretos em um servidor, permitindo que vários serviços funcionem simultaneamente no mesmo endereço IP. Portas garantem que os dados sejam entregues ao aplicativo de destino correto, possibilitando a comunicação entre clientes e serviços específicos.

c) Para que servem as classes de entrada e saída `ObjectInputStream` e `ObjectOutputStream`, e por que os objetos transmitidos devem ser serializáveis?

`ObjectOutputStream`: Permite que objetos sejam convertidos em uma sequência de bytes, tornando-os adequados para transmissão em rede.

`ObjectInputStream`: Realiza a operação oposta, convertendo a sequência de bytes recebida de volta para objetos.

Os objetos transmitidos devem ser serializáveis porque a serialização garante que os objetos sejam convertidos em um formato padronizado de bytes, que pode ser transmitido pela rede e reconstruído em objetos idênticos do lado receptor.

d) Por que, mesmo utilizando as classes de entidades JPA no cliente, foi possível garantir o isolamento do acesso ao banco de dados?

O isolamento do acesso ao banco de dados é alcançado usando o padrão de arquitetura Cliente-Servidor e a abstração fornecida pelas classes de entidades JPA. As classes de entidades JPA permitem que o cliente interaja com os dados de forma orientada a objetos, enquanto a lógica de acesso ao banco de dados é tratada no servidor.

2º Procedimento – Alimentando a Base

CadastroThreadV2:

```
5 package cadastroserver;
6
7 import controller.MovimentosJpaController;
8 import controller.PessoasJpaController;
9 import controller.ProdutosJpaController;
10 import controller.UsuariosJpaController;
11 import java.io.IOException;
12 import java.io.ObjectInputStream;
13 import java.io.ObjectOutputStream;
14 import java.net.Socket;
15 import java.text.SimpleDateFormat;
16 import java.util.Date;
17 import java.util.List;
18 import java.util.logging.Level;
19 import java.util.logging.Logger;
20 import model.Movimentos;
21 import model.Produtos;
22 import model.Pessoas;
23 import model.Usuarios;
24
25 /**
26  *
27  * @author leosc
28  */
29 public class CadastroThreadV2 extends Thread {
30     private final UsuariosJpaController ctrlUsu;
31     private final MovimentosJpaController ctrlMov;
32     private final ProdutosJpaController ctrlProd;
33     private final PessoasJpaController ctrlPessoas;
34     private Usuarios usuarioAutenticado;
35     private final Socket s1;
```

```

36
37
38 public CadastroThreadV2 (UsuariosJpaController ctrlUsu, MovimentosJpaController ctrlMov, ProdutosJpaController ctrlProd, PessoasJpaController ctrlPessoa) {
39     this.ctrlUsu = ctrlUsu;
40     this.ctrlMov = ctrlMov;
41     this.ctrlProd = ctrlProd;
42     this.ctrlPessoa = ctrlPessoa;
43     this.sl = sl;
44 }
45
46 @Override
47 public void run() {
48     String mensagemAutenticacao = "";
49     boolean menu = true;
50     try {
51         ObjectOutputStream out = new ObjectOutputStream(out: sl.getOutputStream());
52         ObjectInputStream in = new ObjectInputStream(in: sl.getInputStream()); {
53
54             String login = (String) in.readObject();
55             String senha = (String) in.readObject();
56
57             List<Usuarios> usuariosList = ctrlUsu.findUsuariosEntities();
58             for (Usuarios usuario : usuariosList) {
59                 if (usuario.getLogin().equals(mensagemAutenticacao: login) && usuario.getSenha().equals(mensagemAutenticacao: senha)) {
60                     usuarioAutenticado = usuario;
61                     mensagemAutenticacao = "Usuario conectado com sucesso";
62                     break;
63                 }
64             }
65
66             if (usuarioAutenticado == null) {
67                 System.out.println("Credenciais inválidas. Desconectando cliente.");
68                 mensagemAutenticacao = "Credenciais inválidas. Desconectando cliente.";
69                 menu = false;
70             }
71
72             String dataHora = new SimpleDateFormat(pattern: "E MMM dd HH:mm:ss z yyyy").format(new Date()); // Obtém a data e hora formatada
73             String mensagemCompleta = ">>> Nova comunicação em " + dataHora ;
74
75             out.writeObject(mensagemCompleta);
76             out.writeObject(mensagemAutenticacao);
77             out.flush();
78
79             while (menu == true) {
80                 String comando = (String) in.readObject();
81                 switch (comando) {
82                     case "E":
83                         processEntrada(in, out);
84                         break;
85                     case "S":
86                         processSaida(in, out);
87                         break;
88                     case "I":
89                         List<Produtos> produtosList = ctrlProd.findProdutosEntities();
90                         dataHora = new SimpleDateFormat(pattern: "E MMM dd HH:mm:ss z yyyy").format(new Date());
91                         mensagemCompleta = ">>> Nova comunicação em " + dataHora ;
92                         out.writeObject(mensagemCompleta);
93                         for (Produtos produto : produtosList) {
94                             String produtoInfo = produto.getNome() + " : " + produto.getQuantidade();
95                             out.writeObject(mensagemCompleta: produtoInfo);
96                             out.flush();
97                         }
98                         break;
99                     case "X":
100                         menu = false;
101                         break;
102                     default:
103                         break;
104                 }
105             }
106         } catch (IOException ex) {
107             System.err.println("Erro de comunicação: " + ex.getMessage());
108         } catch (ClassNotFoundException ex) {
109             Logger.getLogger(CadastroThreadV2.class.getName()).log(Level.SEVERE, msg: null, thrown: ex);
110         } catch (Exception ex) {
111             Logger.getLogger(CadastroThreadV2.class.getName()).log(Level.SEVERE, msg: null, thrown: ex);
112         }
113     }
114 }
115
116 private void processEntrada(ObjectInputStream in, ObjectOutputStream out) throws IOException, ClassNotFoundException, Exception {
117     char tipoMovimento = 'E'; // Tipo de movimento Entrada
118
119     int idPessoa = in.readInt();
120     int idProduto = in.readInt();
121     int quantidade = in.readInt();
122     float valorUnitario = in.readFloat();
123     int idUsuario = usuarioAutenticado.getIdUsuario();
124
125     Usuarios usuario = ctrlUsu.findUsuarios(id: idUsuario);
126     Pessoas pessoa = ctrlPessoa.findPessoas(id: idPessoa);
127     Produtos produto = ctrlProd.findProdutos(id: idProduto);
128
129     Movimentos movimento = new Movimentos();
130     movimento.setIdMovimentos(idMovimentos: getNextMovimentoId());
131     movimento.setIdUsuario(idUsuario: usuario);
132     movimento.setTipo(tipo: tipoMovimento);
133     movimento.setIdPessoa(idPessoa: pessoa);
134     movimento.setIdProduto(idProduto: produto);
135     movimento.setQuantidade(quantidade);
136     movimento.setPrecoUnitario(precoUnitario: valorUnitario);
137
138     ctrlMov.create(movimentos: movimento);
139
140     produto.setQuantidade(produto.getQuantidade() + quantidade);
141     ctrlProd.edit(produtos: produto);
142
143     out.writeObject(mensagemCompleta: "Movimento de Entrada registrado com sucesso.");
144 }
145
146 private void processSaida(ObjectInputStream in, ObjectOutputStream out) throws IOException, ClassNotFoundException, Exception {
147     char tipoMovimento = 'S'; // Tipo de movimento Saida
148
149     int idPessoa = in.readInt();
150     int idProduto = in.readInt();
151     int quantidade = in.readInt();
152     float valorUnitario = in.readFloat();
153     int idUsuario = usuarioAutenticado.getIdUsuario();

```

```

152     Usuarios usuario = ctrlUsu.findUsuarios(id: idUsuario);
153     Pessoas pessoa = ctrlPessoa.findPessoas(id: idPessoa);
154     Produtos produto = ctrlProd.findProdutos(id: idProduto);
155
156     Movimentos movimento = new Movimentos();
157     movimento.setIdMovimentos(idMovimentos: getNextMovimentoId());
158     movimento.setIdUsuario(idUsuario: usuario);
159     movimento.setTipo(tipo: tipoMovimento);
160     movimento.setIdPessoa(idPessoa: pessoa);
161     movimento.setIdProduto(idProduto: produto);
162     movimento.setQuantidade(quantidade);
163     movimento.setPrecoUnitario(precoUnitario: valorUnitario);
164
165     ctrlMov.create(movimentos: movimento);
166
167     produto.setQuantidade(produto.getQuantidade() - quantidade);
168     ctrlProd.edit(produtos: produto);
169
170     out.writeObject(obj: "Movimento de Saída registrado com sucesso.");
171 }
172
173 private synchronized int getNextMovimentoId() {
174     List<Movimentos> movimentos = ctrlMov.findMovimentosEntities();
175     int lastMovimentoId = 0;
176
177     for (Movimentos movimento : movimentos) {
178         if (movimento.getIdMovimentos() > lastMovimentoId) {
179             lastMovimentoId = movimento.getIdMovimentos();
180         }
181     }
182
183     return lastMovimentoId + 1;
184 }
185
186

```

CadastroClientV2:

```

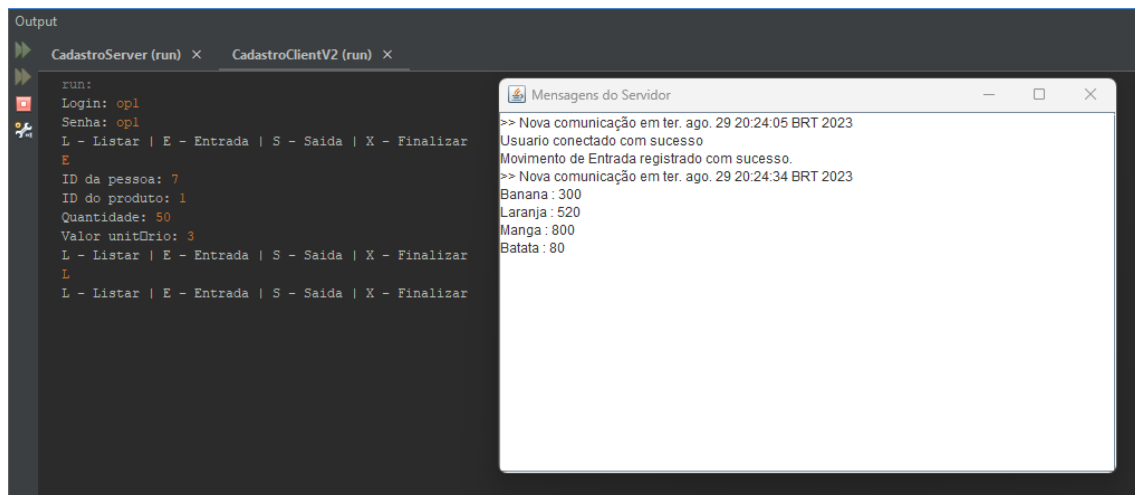
5 package cadastroclientv2;
6
7 import java.io.*;
8 import java.net.*;
9 import javax.swing.*;
10
11 /**
12  * @author leosc
13  */
14 public class CadastroClientV2 {
15
16     /**
17      * @param args the command line arguments
18      */
19     public static void main(String[] args) throws IOException {
20         Socket socket = new Socket(host: "localhost", port: 4321);
21         ObjectOutputStream out = new ObjectOutputStream(out: socket.getOutputStream());
22         ObjectInputStream in = new ObjectInputStream(in: socket.getInputStream());
23         BufferedReader reader = new BufferedReader(new InputStreamReader(in: System.in));
24         boolean menu = true;
25
26         System.out.print(s: "Login: ");
27         String login = reader.readLine();
28         System.out.print(s: "Senha: ");
29         String senha = reader.readLine();
30
31         out.writeObject(obj: login);
32         out.writeObject(obj: senha);
33         out.writeObject(obj: "Mensagem do servidor para o cliente.");
34         out.flush();
35
36         JFrame frame = new JFrame(title: "Mensagens do Servidor");
37         JTextArea textArea = new JTextArea(rows: 20, columns: 50);
38         textArea.setEditable(b: false);
39         frame.add(new JScrollPane(view: textArea));
40         frame.pack();
41         frame.setDefaultCloseOperation(operation: JFrame.EXIT_ON_CLOSE);
42         frame.setVisible(b: true);
43
44         ThreadClient threadClient = new ThreadClient(in, textArea);
45         threadClient.start();
46
47         while (menu == true) {
48             System.out.println(x: "L - Listar | E - Entrada | S - Saída | X - Finalizar");
49             String opcao = reader.readLine().toUpperCase();
50             out.writeObject(obj: opcao);
51             out.flush();
52
53             switch (opcao) {
54                 case "X":
55                     menu = false;
56                     break;
57                 case "L":
58                     break;
59                 case "E":
60                 case "S":
61                     System.out.print(s: "ID da pessoa: ");
62                     int idPessoa = Integer.parseInt(s: reader.readLine());
63
64                     System.out.print(s: "ID do produto: ");
65                     int idProduto = Integer.parseInt(s: reader.readLine());
66
67                     System.out.print(s: "Quantidade: ");
68                     int quantidade = Integer.parseInt(s: reader.readLine());
69
70                     System.out.print(s: "Valor unitário: ");
71                     float valorUnitario = Float.parseFloat(s: reader.readLine());
72
73                     out.writeInt(val: idPessoa);
74                     out.writeInt(val: idProduto);
75                     out.writeInt(val: quantidade);
76                     out.writeFloat(val: valorUnitario);
77                     out.flush();
78                     break;
79                 default:
80                     System.out.println(x: "Opção inválida.");
81                     break;
82             }
83         }
84     }
85 }

```

ThreadClient:

```
5 package cadastroclientv2;
6
7 import java.io.ObjectInputStream;
8 import javax.swing.JTextArea;
9 import javax.swing.SwingUtilities;
10
11 /**
12  *
13  * @author leosc
14  */
15 public class ThreadClient extends Thread {
16     private ObjectInputStream in;
17     private final JTextArea textArea;
18
19     public ThreadClient(ObjectInputStream in, JTextArea textArea) {
20         this.in = in;
21         this.textArea = textArea;
22     }
23
24     @Override
25     public void run() {
26         try {
27             while (true) {
28                 Object data = in.readObject();
29                 String mensagem = (String) data;
30                 SwingUtilities.invokeLater(() -> {
31                     textArea.append(mensagem + "\n");
32                     textArea.setCaretPosition(textArea.getDocument().getLength());
33                 });
34             }
35         } catch (Exception e) {
36
37         }
38     }
39 }
```

Resultado da execução:



- a) Como as Threads podem ser utilizadas para o tratamento assíncrono das respostas enviadas pelo servidor?

Através de uma Thread dedicada à comunicação com o servidor, o cliente pode continuar sua execução normal enquanto aguarda respostas. Isso mantém a interface gráfica responsiva, evitando bloqueios. Ao receber respostas, a Thread assíncrona atualiza a interface de usuário. Isso permite uma melhor experiência

do usuário, pois as operações de comunicação não interferem na interatividade da interface.

- b) Para que serve o método `invokeLater`, da classe `SwingUtilities`?

O método `invokeLater` da classe `SwingUtilities` é usado para executar uma determinada ação de forma assíncrona na thread de eventos do Swing, que é responsável pela atualização da interface gráfica. Isso é essencial para garantir que as atualizações na interface ocorram de forma segura, evitando conflitos entre threads e mantendo a responsividade da aplicação.

- c) Como os objetos são enviados e recebidos pelo Socket Java?

Em Java, objetos são enviados e recebidos através de sockets utilizando a serialização. A serialização converte os objetos em uma sequência de bytes que podem ser transmitidos pela rede. O `ObjectOutputStream` é usado para escrever objetos em bytes, que são enviados pelo socket. O `ObjectInputStream` é usado para ler os bytes recebidos e reconstruir os objetos originais. A classe dos objetos deve implementar a interface `Serializable` para permitir a serialização e desserialização.

- d) Compare a utilização de comportamento assíncrono ou síncrono nos clientes com Socket Java, ressaltando as características relacionadas ao bloqueio do processamento.

Na utilização de sockets Java, o comportamento assíncrono permite que os clientes continuem executando outras tarefas enquanto esperam por respostas do servidor. Isso evita bloqueios e mantém a responsividade do programa. Por outro lado, o comportamento síncrono exige que o cliente aguarde a resposta do servidor antes de continuar, o que pode resultar em bloqueios e tornar o programa menos eficiente em termos de utilização de recursos e tempo de resposta. O comportamento assíncrono é preferível para garantir a fluidez da interação do usuário e otimizar o uso de recursos do sistema.

Conclusão

Desenvolvemos uma aplicação cliente-servidor para gerenciamento de cadastros, implementação robusta das funcionalidades de autenticação, entradas e saídas, organização eficiente com classes de controle separadas, utilização adequada de threads para comunicação assíncrona e criação de interface gráfica com a biblioteca Swing. Isso proporcionou uma valiosa experiência na aplicação de conceitos teóricos em um cenário prático.