



DESENVOLVIMENTO FULL STACK
POLO SETOR O – CEILÂNDIA - DF
PERÍODO 2024.2 FLEX
DISCIPLINA: PORQUE NÃO PARALELIZAR?
MAYCON MOURA

Título da Prática

Utilização de Threads e Comunicação Assíncrona com Socket Java para Gerenciamento de Respostas de Servidor

Objetivo da Prática

O objetivo desta prática é explorar o uso de Threads para manipular a comunicação assíncrona em uma aplicação cliente-servidor utilizando Sockets em Java. Além disso, será analisada a utilização do método `invokeLater` da classe `SwingUtilities` para garantir a atualização correta da interface gráfica, assim como a forma como os objetos são enviados e recebidos via Sockets em Java. A prática também abordará uma comparação entre comportamento assíncrono e síncrono no contexto da comunicação via Socket.

Códigos

CadastroClienteV2.java

```
/*  
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to  
change this license  
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this  
template  
*/
```

```

package cadastrserver;

import cadastrserver.model.Produto;
import java.awt.BorderLayout;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.SwingUtilities;

/**
 *
 * @author maycon
 */
public class CadastroClientV2 {

    public static void main(String[] args) {
        String host = "localhost";
        int porta = 4321;

        try {
            System.out.println("Tentando conectar ao servidor...");
            Socket socket = new Socket(host, porta);
            System.out.println("Conectado ao servidor em " + host + ":" + porta);

            ObjectOutputStream out = new
ObjectOutputStream(socket.getOutputStream());
            ObjectInputStream in = new ObjectInputStream(socket.getInputStream());

            String login = "op1";
            String senha = "op1";
            out.writeObject(login);
            out.writeObject(senha);
            out.flush();
            System.out.println("Login e senha enviados.");

            Object respostaLogin = in.readObject();
            System.out.println("Resposta do login: " + respostaLogin);

```

```

        if (respostaLogin instanceof String && "Credenciais inválidas. Conexão
encerrada.".equals(respostaLogin)) {
            System.out.println("Login ou senha inválidos. Encerrando conexão.");
            socket.close();
            return;
        }

```

```

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

```

```

        SaidaFrame saidaFrame = new SaidaFrame(null);
        saidaFrame.setVisible(true);

```

```

        ThreadClient threadClient = new ThreadClient(in, saidaFrame.texto);
        threadClient.start();

```

```

        String comando = "";

```

```

        while (!comando.equalsIgnoreCase("X")) {
            System.out.println("\nMenu:");
            System.out.println("L – Listar");
            System.out.println("E – Entrada");
            System.out.println("S – Saída");
            System.out.println("X – Finalizar");
            System.out.print("Escolha uma opção: ");

```

```

            comando = teclado.readLine().toUpperCase();
            System.out.println("Comando recebido: " + comando);

```

```

            switch (comando) {
                case "L":
                    out.writeObject("L");
                    out.flush();
                    System.out.println("Comando 'L' enviado.");
                    break;

```

```

                case "E":
                case "S":
                    out.writeObject(comando);
                    out.flush();

```

```

                    System.out.print("Digite o Id da pessoa: ");
                    int pessoald = Integer.parseInt(teclado.readLine());
                    out.writeObject(pessoald);
                    out.flush();
                    System.out.println("Id da pessoa enviado: " + pessoald);

```

```

                    System.out.print("Digite o Id do produto: ");

```

```

        int produtold = Integer.parseInt(teclado.readLine());
        out.writeObject(produtold);
        out.flush();
        System.out.println("Id do produto enviado: " + produtold);

        System.out.print("Digite a quantidade: ");
        int quantidade = Integer.parseInt(teclado.readLine());
        out.writeObject(quantidade);
        out.flush();
        System.out.println("Quantidade enviada: " + quantidade);

        System.out.print("Digite o valor unitário: ");
        double valorUnitario = Double.parseDouble(teclado.readLine());
        out.writeObject(valorUnitario);
        out.flush();
        System.out.println("Valor unitário enviado: " + valorUnitario);

        break;

    case "X":
        System.out.println("Finalizando a conexão.");
        threadClient.interrupt();
        out.writeObject("X");
        out.flush();
        socket.close();
        break;

    default:
        System.out.println("Comando desconhecido.");
    }
}

} catch (Exception e) {
    e.printStackTrace();
}
}

/**
 * Classe interna para gerenciar a leitura de mensagens do servidor e
 * atualizar o JTextArea.
 */
static class ThreadClient extends Thread {

    private ObjectInputStream entrada;
    private JTextArea textArea;

    public ThreadClient(ObjectInputStream entrada, JTextArea textArea) {
        this.entrada = entrada;
    }

```

```

        this.textArea = textArea;
    }

    @Override
    public void run() {
        try {
            while (!Thread.currentThread().isInterrupted()) {
                try {
                    Object mensagem = entrada.readObject();

                    SwingUtilities.invokeLater(() -> {
                        if (mensagem instanceof String) {
                            textArea.append((String) mensagem + "\n");
                        } else if (mensagem instanceof List<?>) {
                            @SuppressWarnings("unchecked")
                            List<?> lista = (List<?>) mensagem;
                            for (Object item : lista) {
                                if (item instanceof Produto) {
                                    Produto produto = (Produto) item;
                                    textArea.append(String.format("Produto: %s, Quantidade: %d\n", produto.getNome(), produto.getQuantidade()));
                                }
                            }
                        } else {
                            textArea.append("Objeto desconhecido recebido.\n");
                        }
                    });

                    try {
                        Thread.sleep(100);
                    } catch (InterruptedException ex) {
                        Logger.getLogger(CadastroClientV2.class.getName()).log(Level.SEVERE, null, ex);
                    }
                } catch (ClassNotFoundException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();
                    Thread.currentThread().interrupt();
                }
            }
        } finally {
            SwingUtilities.invokeLater(() -> textArea.append("Conexão com o servidor encerrada.\n"));
        }
    }
}

```

```

static class SaidaFrame extends JDialog {

    public JTextArea texto;

    public SaidaFrame(JFrame owner) {
        super(owner, "Saída de Mensagens", false);

        texto = new JTextArea();
        texto.setEditable(false);

        texto.setLineWrap(true);
        texto.setWrapStyleWord(true);

        JScrollPane scrollPane = new JScrollPane(texto);
        setLayout(new BorderLayout());
        add(scrollPane, BorderLayout.CENTER);

        setSize(400, 300);

        setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);

        setLocationRelativeTo(null);
    }
}

```

CadastroThread2.java

```

/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to
change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this
template
 */
package cadastroserver;

import cadastroserver.model.Movimento;
import cadastroserver.model.Pessoa;
import cadastroserver.model.Produto;
import cadastroserver.model.Usuario;
import controller.MovimentoJpaController;
import controller.PessoaJpaController;
import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

```

```

import java.math.BigDecimal;
import java.net.Socket;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author maycon
 */
public class CadastroThread2 extends Thread {

    private ProdutoJpaController ctrlProd;
    private UsuarioJpaController ctrlUsu;
    private MovimentoJpaController ctrlMov;
    private PessoaJpaController ctrlPessoa;
    private Socket s1;
    private Usuario usuarioLogado;

    public CadastroThread2(ProdutoJpaController ctrlProd, UsuarioJpaController
ctrlUsu, MovimentoJpaController ctrlMov, PessoaJpaController ctrlPessoa, Socket
s1) {
        this.ctrlProd = ctrlProd;
        this.ctrlUsu = ctrlUsu;
        this.ctrlMov = ctrlMov;
        this.ctrlPessoa = ctrlPessoa;
        this.s1 = s1;
    }

    @Override
    public void run() {
        try {
            ObjectOutputStream out = new ObjectOutputStream(s1.getOutputStream());
            ObjectInputStream in = new ObjectInputStream(s1.getInputStream());

            // Autenticação do Usuário
            String login = (String) in.readObject();
            String senha = (String) in.readObject();
            System.out.println("Recebido login: " + login);
            System.out.println("Recebida senha: " + senha);
            usuarioLogado = ctrlUsu.findUsuario(login, senha);

            if (usuarioLogado == null) {
                out.writeObject("Credenciais inválidas. Conexão encerrada.");
                out.flush(); // Garante que a mensagem seja enviada imediatamente
                s1.close();
                return;
            }
        }
    }
}

```

```

        out.writeObject("Login bem-sucedido. Envie 'L' para listar os produtos.");
        out.flush(); // Garante que a mensagem seja enviada imediatamente

        // Loop de resposta
        boolean continuar = true;
        while (continuar) {
            String comando = (String) in.readObject();
            System.out.println("Comando recebido: " + comando);

            switch (comando) {
                case "L":
                    // Listar produtos
                    List<Produto> produtos = ctrlProd.findProdutoEntities();
                    out.writeObject(produtos);
                    break;

                case "E":
                case "S":
                    // Entrada ou saída de produtos
                    processarMovimento(comando.charAt(0), in, out);
                    break;

                case "X":
                    continuar = false;
                    break;

                default:
                    out.writeObject("Comando desconhecido.");
            }
            out.flush(); // Garante que as respostas sejam enviadas imediatamente
        }

        s1.close();

    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

```

private void processarMovimento(char tipoMovimento, ObjectInputStream in,
ObjectOutputStream out) throws IOException, ClassNotFoundException {
    // Receber dados do movimento
    int pessoald = (int) in.readObject();
    int produtold = (int) in.readObject();
    int quantidade = (int) in.readObject();
    double valorUnitario = (double) in.readObject();
}

```



```

// Criar um novo movimento
Movimento movimento = new Movimento();
movimento.setIdUsuario(usuarioLogado);
movimento.setTipo(tipoMovimento);
Pessoa pessoa = new Pessoa(pessoald);
movimento.setIdPessoa(pessoa);
movimento.setIdProduto(ctrlProd.findProduto(produtold));
movimento.setQuantidade(quantidade);
movimento.setPrecoUnitario(BigDecimal.valueOf(valorUnitario));

// Persistir o movimento
ctrlMov.create(movimento);

// Atualizar a quantidade do produto
Produto produto = ctrlProd.findProduto(produtold);
if (produto != null) {
    try {
        int novaQuantidade = tipoMovimento == 'E' ?
            produto.getQuantidade() + quantidade :
            produto.getQuantidade() - quantidade;

        produto.setQuantidade(novaQuantidade);
        ctrlProd.edit(produto);

        out.writeObject("Movimento registrado com sucesso.");
    } catch (Exception ex) {
        Logger.getLogger(CadastroThread2.class.getName()).log(Level.SEVERE,
null, ex);
        out.writeObject("Erro ao registrar movimento.");
    }
} else {
    out.writeObject("Produto não encontrado.");
}
}
}

```

CadastroServer.java

```

/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to
change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this
template
 */
package cadastroserver;

import controller.MovimentoJpaController;
import controller.PessoaJpaController;

```

```

import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.HashMap;
import java.util.Map;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

/**
 *
 * @author maycon
 */
public class CadastroServer {

    private static final Map<String, String> credenciais = new HashMap<>();

    static {
        credenciais.put("usuario1", "senha1");
        credenciais.put("usuario2", "senha2");
    }

    public static void main(String[] args) {
        EntityManagerFactory emf =
Persistence.createEntityManagerFactory("CadastroServerPU");

        ProdutoJpaController ctrlProd = new ProdutoJpaController(emf);
        UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);
        MovimentoJpaController ctrlMov = new MovimentoJpaController(emf);
        PessoaJpaController ctrlPessoa = new PessoaJpaController(emf);

        int porta = 4321;

        try {
            ServerSocket serverSocket = new ServerSocket(porta);
            System.out.println("Servidor escutando na porta " + porta);

            while (true) {
                Socket socket = serverSocket.accept();
                System.out.println("Novo cliente conectado!");

                CadastroThread2 thread = new CadastroThread2(ctrlProd, ctrlUsu,
ctrlMov, ctrlPessoa, socket);
                thread.start();
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```

    }

    /*try (ServerSocket serverSocket = new ServerSocket(12345)) {
        System.out.println("Servidor iniciado. Aguardando conexão...");

        while (true) {
            try (Socket clientSocket = serverSocket.accept()) {
                System.out.println("Cliente conectado.");

                BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
                PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

                out.println("Digite seu login:");
                String login = in.readLine();
                out.println("Digite sua senha:");
                String senha = in.readLine();

                if (validaCredenciais(login, senha)) {
                    out.println("Login bem-sucedido. Envie 'L' para listar os produtos.");
                } else {
                    out.println("Credenciais inválidas. Conexão encerrada.");
                    continue;
                }

                String comando;
                while ((comando = in.readLine()) != null) {
                    if (comando.equalsIgnoreCase("L")) {
                        out.println(listaProdutos());
                    } else {
                        out.println("Comando não reconhecido.");
                    }
                }
            } catch (IOException e) {
                System.out.println("Erro na conexão com o cliente: " + e.getMessage());
            }
        }
    } catch (IOException e) {
        System.err.println("Erro ao iniciar o servidor: " + e.getMessage());
    }
    */
}

private static boolean validaCredenciais(String login, String senha) {
    return credenciais.containsKey(login) && credenciais.get(login).equals(senha);
}

// Método que retorna a lista de produtos (exemplo)
private static String listaProdutos() {
    return "1 - Produto A\n2 - Produto B\n3 - Produto C";
}

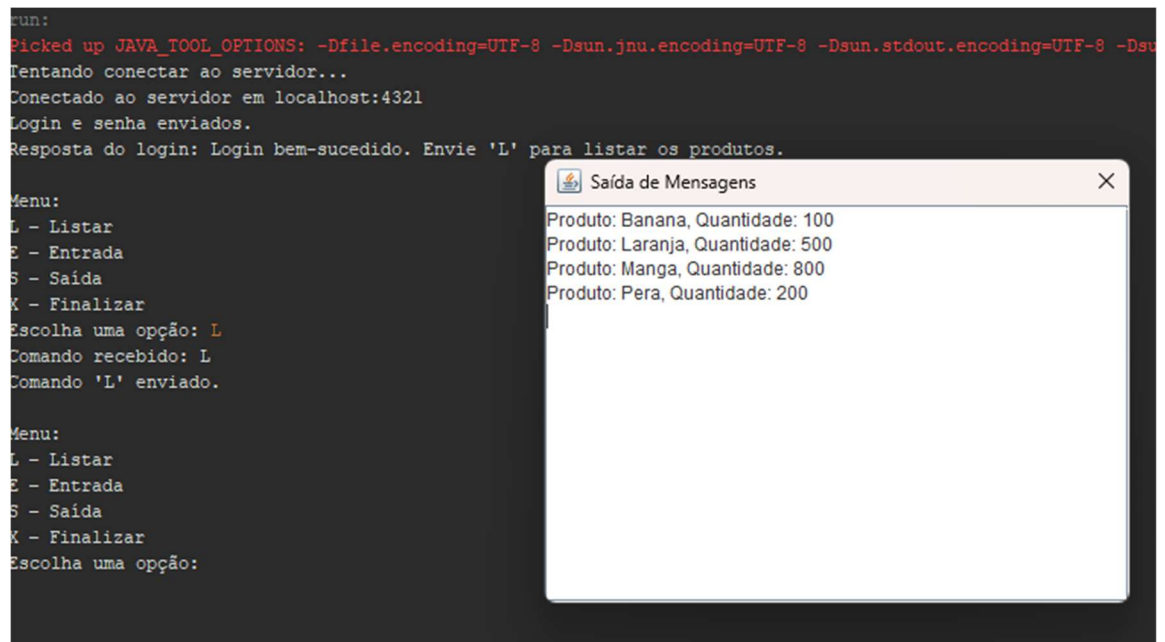
```

```
}  
  
}
```

Resultados

```
Tentando conectar ao servidor...  
Conectado ao servidor em localhost:4321  
Login e senha enviados.  
Resposta do login: Login bem-sucedido. Envie 'L' para listar os produtos.  
  
Menu:  
L - Listar  
E - Entrada  
S - Saida  
X - Finalizar  
Escolha uma opção: E  
Comando recebido: E  
Digite o Id da pessoa: 1  
Id da pessoa enviado: 1  
Digite o Id do produto: 1  
Id do produto enviado: 1  
Digite a quantidade: 2  
Quantidade enviada: 2  
Digite o valor unitário: 5  
Valor unitário enviado: 5.0  
  
Menu:  
L - Listar  
E - Entrada  
S - Saida  
X - Finalizar  
Escolha uma opção:
```

Cliente



The screenshot shows a Java application running in a terminal window. The terminal output is as follows:

```
run:
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF-8 -Dsun.jnu.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8
Tentando conectar ao servidor...
Conectado ao servidor em localhost:4321
Login e senha enviados.
Resposta do login: Login bem-sucedido. Envie 'L' para listar os produtos.

Menu:
L - Listar
E - Entrada
S - Saída
X - Finalizar
Escolha uma opção: L
Comando recebido: L
Comando 'L' enviado.

Menu:
L - Listar
E - Entrada
S - Saída
X - Finalizar
Escolha uma opção:
```

Overlaid on the terminal is a small Java Swing window titled "Saída de Mensagens". It contains a list of products and their quantities:

- Produto: Banana, Quantidade: 100
- Produto: Laranja, Quantidade: 500
- Produto: Manga, Quantidade: 800
- Produto: Pera, Quantidade: 200

Análise e Conclusão

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

As Threads são essenciais no tratamento assíncrono de respostas enviadas pelo servidor em uma aplicação cliente-servidor. Ao utilizar uma Thread separada para gerenciar a comunicação com o servidor, é possível continuar a execução do programa sem bloquear a interface gráfica ou o processamento de outras tarefas.

No exemplo da aplicação desenvolvida, uma Thread foi criada especificamente para ficar aguardando respostas do servidor (Thread de leitura). Essa abordagem permite que, enquanto o cliente realiza outras atividades, como a interação com a interface gráfica, as mensagens do servidor possam ser recebidas e processadas em paralelo.

A Thread de leitura funciona como um ouvinte que fica em "loop", aguardando novas mensagens do servidor e, assim que recebe uma, ela processa essa mensagem sem interferir na execução de outras partes do sistema. Isso evita que o programa seja bloqueado esperando uma resposta do servidor, o que caracteriza um comportamento assíncrono.

2. Para que serve o método `invokeLater`, da classe `SwingUtilities`?

O método `SwingUtilities.invokeLater` é usado para garantir que atualizações na interface gráfica sejam realizadas na Event Dispatch Thread (EDT), a thread responsável por manipular todos os eventos e atualizações da interface gráfica no Swing. Isso é

importante porque o Swing não é thread-safe, ou seja, atualizar a interface gráfica de outras threads pode resultar em comportamento imprevisível.

No exemplo da aplicação, as mensagens recebidas do servidor pela Thread de leitura são manipuladas dentro dessa Thread. Porém, para garantir que o JTextArea seja atualizado corretamente, utilizamos `invokeLater`, que posta o trabalho de atualização da interface gráfica na EDT. Isso garante que o Swing manipule a interface de forma adequada e segura.

3. Como os objetos são enviados e recebidos pelo Socket Java?

Em Java, objetos podem ser enviados e recebidos através de Sockets usando streams de objetos, especificamente os objetos `ObjectOutputStream` e `ObjectInputStream`. Esses streams permitem que objetos serializáveis sejam transmitidos pela rede como uma sequência de bytes e reconstruídos no outro lado da conexão.

Para enviar um objeto, usamos o método `writeObject(Object obj)` de um `ObjectOutputStream`, e para receber, usamos o método `readObject()` de um `ObjectInputStream`. É importante garantir que os objetos envolvidos implementem a interface `Serializable`, pois isso permite que o Java converta o objeto em uma forma que possa ser transmitida pela rede.

No exemplo da aplicação, o cliente envia comandos, IDs e quantidades ao servidor usando o `writeObject()`, e o servidor responde com mensagens que são lidas pelo `readObject()`. O mesmo mecanismo é usado para transmitir listas de objetos, como listas de produtos.

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

A principal diferença entre o comportamento assíncrono e síncrono em uma comunicação cliente-servidor com Socket Java está no controle do fluxo e no impacto no desempenho da aplicação.

Síncrono: No modelo síncrono, o cliente envia uma requisição ao servidor e aguarda, de forma bloqueante, até que o servidor responda. Isso significa que enquanto o cliente aguarda a resposta, ele não pode realizar outras atividades, resultando em uma parada no fluxo de execução até que o servidor conclua o processamento e envie a resposta. Esse comportamento pode ser simples de implementar, mas é ineficiente em casos em que o tempo de resposta do servidor é imprevisível ou longo, pois o cliente fica ocioso, aguardando o término da operação.

Assíncrono: No modelo assíncrono, o cliente pode enviar uma requisição e continuar executando outras tarefas enquanto espera pela resposta. Uma Thread separada pode ser utilizada para aguardar e processar as respostas do servidor, sem bloquear a execução do programa. Isso melhora a responsividade da aplicação, principalmente em cenários com interface gráfica ou com operações que requerem tempo de espera.

prolongado. O comportamento assíncrono permite maior flexibilidade, uma vez que o cliente não precisa "parar" enquanto aguarda as respostas.

Em nossa prática, o uso de uma Thread dedicada para a leitura de respostas do servidor ilustra bem o comportamento assíncrono. Se o comportamento fosse síncrono, o programa teria que aguardar a resposta do servidor antes de continuar a execução, o que não é desejável em muitos casos, especialmente em sistemas com interface gráfica.

Conclusão Final

A utilização de Threads e de comunicação assíncrona com Sockets em Java é uma prática recomendada para melhorar a performance e a usabilidade de aplicações cliente-servidor, especialmente aquelas que possuem interfaces gráficas ou que realizam operações demoradas. O uso do método `invokeLater` em conjunto com Threads permite garantir que a interface gráfica seja atualizada corretamente e de maneira thread-safe, evitando problemas comuns de concorrência em aplicativos Swing. A comparação entre comportamentos assíncronos e síncronos mostra que, para sistemas interativos, o modelo assíncrono é mais eficiente e apropriado.