UNIVERSIDADE ESTACIO DE SÁ DE RIBEIRÃO PRETO

POLO PARQUE ANDORINHAS

DESENVOLVIMENTO FULL STACK

2023.3 FULL STACK ALUNO

NIVEL 5: POR QUE NÃO PARALELIZAR?

ALESSANDRO SENDI SHIGEMATSU

Título da Prática

RPG0018 - Por que não paralelizar

Objetivos da prática

1. Criar servidores Java com base em Sockets.
2. Criar clientes síncronos para servidores com base em Sockets.
3. Criar clientes assíncronos para servidores com base em Sockets.
4. Utilizar Threads para implementação de processos paralelos.
5. No final do exercício, o aluno terá criado um servidor Java baseado em Socket, com acesso ao banco de dados via JPA, além de utilizar os recursos nativos do Java para implementação de clientes síncronos e assíncronos. As Threads serão usadas tanto no servidor, para viabilizar múltiplos clientes paralelos, quanto no cliente, para implementar a resposta assíncrona.

1º Procedimento | Criando o Servidor e Cliente de Teste

Todos os códigos solicitados neste roteiro de aula

Classe CadastroClient

```
package cadastroclient;


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 model.Produto;


public class CadastroClient {
```

```
    public static void main(String[] args) throws IOException, ClassNotFoundException {
        Socket socket = new Socket("localhost", 4321);
        ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
        ObjectInputStream in = new ObjectInputStream(socket.getInputStream());
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        System.out.println("------------------------------");
        System.out.print("Login: ");
        String login = reader.readLine();
        System.out.print("Senha: ");
        String senha = reader.readLine();

        out.writeObject(login);
        out.writeObject(senha);
//      out.writeObject("Mensagem do cliente para o servidor.");
        out.flush();

        String mensagem = (String) in.readObject();
        System.out.println("mensagem recebida do servidor=" + mensagem);

        if( "NAO AUTORIZADO".equals(mensagem) ) return;

        System.out.print("Comando desejado: ");
        String comando = reader.readLine();
        out.writeObject(comando);
        out.flush();

        List<Produto> listaProdutos = (List<Produto>) in.readObject();

        listaProdutos.stream().forEach( produto -> System.out.println( produto.getNome() ));
```

```
        socket.close();
    }

}
```

Classe CadastroThread

```
package cadastroserver;

import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import java.io.IOException;
import java.net.Socket;
import java.util.logging.Level;
import java.util.logging.Logger;
import model.Usuario;
import model.Produto;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.List;
import java.util.stream.Collectors;

public class CadastroThread  extends Thread{
    private final UsuarioJpaController ctrlUsu;
    private final ProdutoJpaController ctrl;
    private final Socket s1;

    public CadastroThread( ProdutoJpaController ctrl ,UsuarioJpaController ctrlUsu, Socket s1) {
        this.ctrl = ctrl;
        this.ctrlUsu = ctrlUsu;
        this.s1 = s1;
    }
```

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

            String login = (String) in.readObject();
            String senha = (String) in.readObject();
//             String mensagem = (String) in.readObject();
            out.flush();

            boolean loginValidado = realizarLogin( login,senha );

            if( loginValidado == false ){
               enviarMensagemCliente( out,"NAO AUTORIZADO" );
               s1.close();
               return;
            }

            enviarMensagemCliente( out,"USUARIO CONECTADO COM SUCESSO" );

            while( true ){
               String comando = (String) in.readObject();
               out.flush();

               switch (comando.toUpperCase()) {
                  case "L" -> listarProdutos( out );
                  default -> enviarMensagemCliente(out, "Comando invalido! Favor tentar
novamente");
               }
            }
```

```
            } catch (IOException ex) {
            try {
                s1.close();
            } catch (IOException ex1) {
                Logger.getLogger(CadastroThread.class.getName()).log(Level.SEVERE, null, ex1);
            }
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(CadastroThread.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    private boolean realizarLogin( String login,String senha ){
        List<Usuario> listaUsuario = ctrlUsu.findUsuarioEntities();
        return listaUsuario.stream().anyMatch( usuario -> login.equals(usuario.getLogin() ) &&
senha.equals(usuario.getSenha()) );
    }

    private void enviarMensagemCliente( ObjectOutputStream out, String mensagem ) throws
IOException{
        out.writeObject(mensagem );
        out.flush();
    }

    private void listarProdutos( ObjectOutputStream out ) throws IOException {

        List<Produto> listaProdutos = ctrl.findProdutoEntities();

        out.writeObject(listaProdutos );

    }
}
```

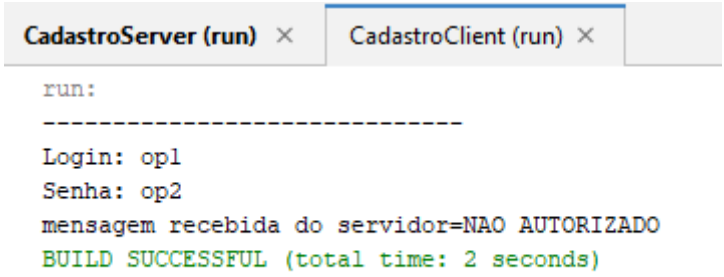
Classe CadastroServer

```
package cadastroserver;

import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import java.io.IOException;

import java.net.ServerSocket;
import java.net.Socket;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;



public class CadastroServer {

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

      ProdutoJpaController ctrl = new ProdutoJpaController(emf);
      UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);

      try (ServerSocket serverSocket = new ServerSocket(4321)) {
         System.out.println("Servidor aguardando conexoes na porta 4321...");
         while (true) {
            Socket socket = serverSocket.accept();



            CadastroThread thread = new CadastroThread(ctrl ,ctrlUsu, socket);
            thread.start();
            System.out.println("thread iniciado!");
```

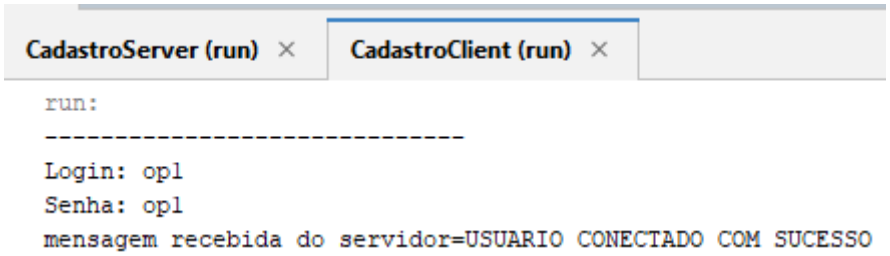
```
        }
         }

    }

}
```

Resultado do login

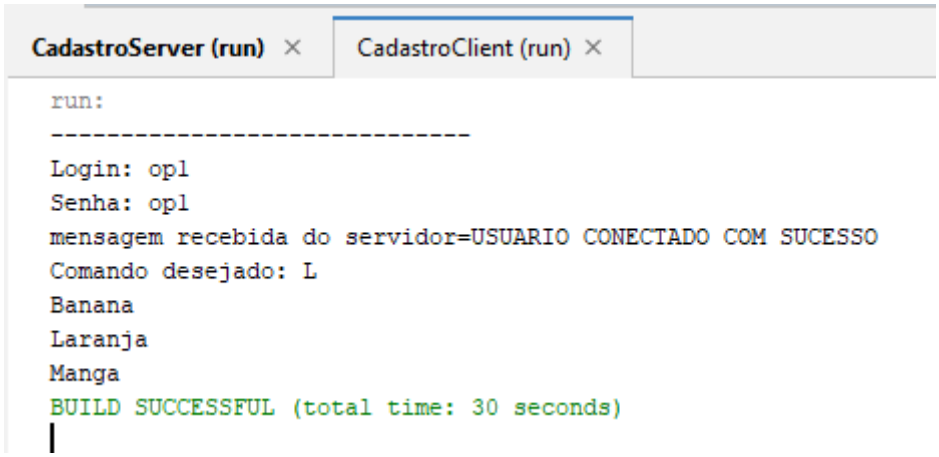
Não sucedido



Sucedido



Listar Produtos



Análise

a) Como funcionam as classes Socket e ServerSocket?

R: Os sockets são vinculados a um número de porta específica, esperando por uma requisição realizada pelo cliente.

O ServerSocket permite que o cliente se conecte ao servidor. Quando o cliente conecta, o server socket cria uma instancia objeto socket, na qual o servidor pode usar para a comunicação com cliente.

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

R:Serve para identificar processos específicos para que os pacotes de dados a chegar sejam facilmente enviados para a aplicação ativa.

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

R: O objectInputStream lê uma sequência de bytes(stream) e converte para objeto java e o ObjectOutputStream converte o objeto serializavel e converte em uma sequência de bytes. Os objetos serializáveis podem ser persistidos ou salvos como objetos,

JVM independente, fácil de entender e modificar, usado para *Marshalling.*

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

R: Porque foi criado instancias diferentes ao acesso do banco de dados através do multithreads.

2º Procedimento | Servidor Completo e Cliente Assíncrono

Todos os códigos solicitados neste roteiro de aula

CLASS CADASTRO SERVER

```
package cadastroserver;


import controller.MovimentoJpaController;
import controller.PessoaFisicaJpaController;
import controller.PessoaJuridicaJpaController;
```

```
import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import java.io.IOException;

import java.net.ServerSocket;
import java.net.Socket;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;



public class CadastroServer {


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


      ProdutoJpaController ctrl = new ProdutoJpaController(emf);
      UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);
      MovimentoJpaController ctrlMov = new MovimentoJpaController(emf);
      PessoaFisicaJpaController ctrlPesFisica = new PessoaFisicaJpaController(emf);
      PessoaJuridicaJpaController ctrlPesJuridica = new PessoaJuridicaJpaController(emf);


      try (ServerSocket serverSocket = new ServerSocket(4321)) {
         System.out.println("Servidor aguardando conexoes na porta 4321...");
         while (true) {
            Socket socket = serverSocket.accept();



//             CadastroThread thread = new CadastroThread(ctrl ,ctrlUsu, socket);
            CadastroThreadV2 thread = new CadastroThreadV2(ctrl ,ctrlUsu,ctrlMov,ctrlPesFisica,
ctrlPesJuridica,socket);
            thread.start();
```

```
            System.out.println("thread iniciado!");

        }
         }

   }

}
```

CLASSE CADASTRO THREAD V2

```
package cadastroserver;

import controller.MovimentoJpaController;
import controller.PessoaFisicaJpaController;
import controller.PessoaJuridicaJpaController;
import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import java.io.IOException;
import java.net.Socket;
import java.util.logging.Level;
import java.util.logging.Logger;
import model.Usuario;
import model.Produto;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.List;
import java.util.stream.Collectors;
import model.Movimento;
import model.PessoaFisica;
import model.PessoaJuridica;

public class CadastroThreadV2 extends Thread {
```

```
    private final UsuarioJpaController ctrlUsu;
    private final ProdutoJpaController ctrl;
    private final MovimentoJpaController ctrlMov;
    private final PessoaFisicaJpaController ctrlPesFisica;
    private final PessoaJuridicaJpaController ctrlPesJuridica;
    private final Socket s1;

    public CadastroThreadV2(ProdutoJpaController ctrl, UsuarioJpaController ctrlUsu,
MovimentoJpaController ctrlMov, PessoaFisicaJpaController ctrlPesFisica,
PessoaJuridicaJpaController ctrlPesJuridica, Socket s1) {
        this.ctrlUsu = ctrlUsu;
        this.ctrl = ctrl;
        this.ctrlMov = ctrlMov;
        this.ctrlPesFisica = ctrlPesFisica;
        this.ctrlPesJuridica = ctrlPesJuridica;
        this.s1 = s1;
    }

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

            String login = (String) in.readObject();
            String senha = (String) in.readObject();
            Usuario usuarioLogado = null;

            try{
                usuarioLogado = BuscarUsuario(login, senha);
```

```
}catch( Exception ex){
   enviarMensagemCliente(out, "NAO AUTORIZADO");
   s1.close();
   return;
}


enviarMensagemCliente(out, "USUARIO CONECTADO COM SUCESSO");

while (true) {
   String comando = (String) in.readObject();
   comando = comando.toUpperCase();

   switch (comando) {
      case "L" ->
         enviarMensagemCliente(out, listarProdutos());
      case "E", "S" -> {

         int idPessoa = (int) in.readObject();
         int idProduto = (int) in.readObject();
         int quantidade = (int) in.readObject();
         long valorUnitario = (long) in.readObject();

         Movimento movimento = new Movimento();

         if( "E".equals(comando) ){
           PessoaJuridica pessoaJuridica = ctrlPesJuridica.findPessoaJuridica(idPessoa);
           movimento.setPessoaidPessoa( pessoaJuridica.getPessoa() );
         }

         if( "S".equals(comando) ){
```

```
                    PessoaFisica pessoaFisica = ctrlPesFisica.findPessoaFisica(idPessoa);
                    movimento.setPessoaidPessoa( pessoaFisica.getPessoa() );
                }

                movimento.setUsuarioidUsuario( usuarioLogado );
                movimento.setProdutoidProduto(ctrl.findProduto(idProduto));
                movimento.setQuantidade(quantidade);
                movimento.setValorUnitario(valorUnitario);
                movimento.setTipo(comando);

                ctrlMov.create(movimento);

    Produto produto = ctrl.findProduto(idProduto);

                if( "E".equals(comando) ) produto.setQuantidade(produto.getQuantidade() +
quantidade);
                if( "S".equals(comando) ) produto.setQuantidade(produto.getQuantidade() -
quantidade);

                ctrl.edit(produto);
                out.writeObject( "E".equals(comando) ? "ENTRADA": "SAIDA" + " registrada com
sucesso!");

            }

            default ->
                enviarMensagemCliente(out, "Comando invalido! Favor tentar novamente");
        }

    }

    } catch (IOException ex) {
```

```
            try {
                System.out.println("ERRO!");
                s1.close();
            } catch (IOException ex1) {
                Logger.getLogger(CadastroThreadV2.class.getName()).log(Level.SEVERE, null, ex1);
            }
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(CadastroThreadV2.class.getName()).log(Level.SEVERE, null, ex);
        } catch (Exception ex) {
            Logger.getLogger(CadastroThreadV2.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    private Usuario BuscarUsuario(String login, String senha) {
        List<Usuario> listaUsuario = ctrlUsu.findUsuarioEntities();
        return listaUsuario.stream().filter(usuario -> login.equals(usuario.getLogin()) &&
senha.equals(usuario.getSenha())).findFirst().get();
    }

    private void enviarMensagemCliente(ObjectOutputStream out, String mensagem) throws
IOException {
        out.writeObject(mensagem);
        out.flush();
    }

    private String listarProdutos() {
        String newLine = System.getProperty("line.separator");

        List<Produto> listaProdutos = ctrl.findProdutoEntities();

        return listaProdutos.stream().map(produto
                -> produto.getNome() + "::" + produto.getQuantidade()
```

```
        ).collect(Collectors.joining(newLine));


    }
}
```

CLASSE CadastroClientV2

```
package cadastroclient;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.math.BigDecimal;
import java.net.Socket;
import java.util.Scanner;

public class CadastroClientV2 {

    public static void main(String[] args) throws IOException, ClassNotFoundException {
        Socket socket = new Socket("localhost", 4321);
        ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
        ObjectInputStream in = new ObjectInputStream(socket.getInputStream());
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        Scanner scan = new Scanner(System.in);


        System.out.println("------------------------------");
        System.out.print("Login: ");
        String login = reader.readLine();
        System.out.print("Senha: ");
        String senha = reader.readLine();
```

```
out.writeObject(login);
out.writeObject(senha);
out.flush();

ThreadClient threadClient = new ThreadClient(in);
threadClient.start();

 while(true)   {
   System.out.print("L – Listar |  X – Finalizar | E– Entrada | S – Saida ");
   String comando = reader.readLine();
   out.writeObject(comando);
   out.flush();

   switch( comando.toUpperCase() ) {
      case "E","S" ->{
         System.out.print("ID da Pessoa:");
         int idPessoa = scan.nextInt();

         System.out.print("Id do Produto:");
         int idProduto = scan.nextInt();

         System.out.print("Quantidade:");
         int quantidade = scan.nextInt();

         System.out.print("Valor Unitario:");
         long valorUnitario = scan.nextLong();

         out.writeObject(idPessoa);
         out.writeObject(idProduto);
         out.writeObject(quantidade);
         out.writeObject(valorUnitario);
```

```
            }
            case "X" -> {
                System.out.print("PROGRAMA FINALIZADO!");
                socket.close();
                System.exit(0);
            }
          }
        }

    }

}
```

CLASSE ThreadClient

```
/*
 * 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 cadastroclient;

import java.io.ObjectInputStream;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.SwingUtilities;

/**
 *
 * @author sendi
 */
```

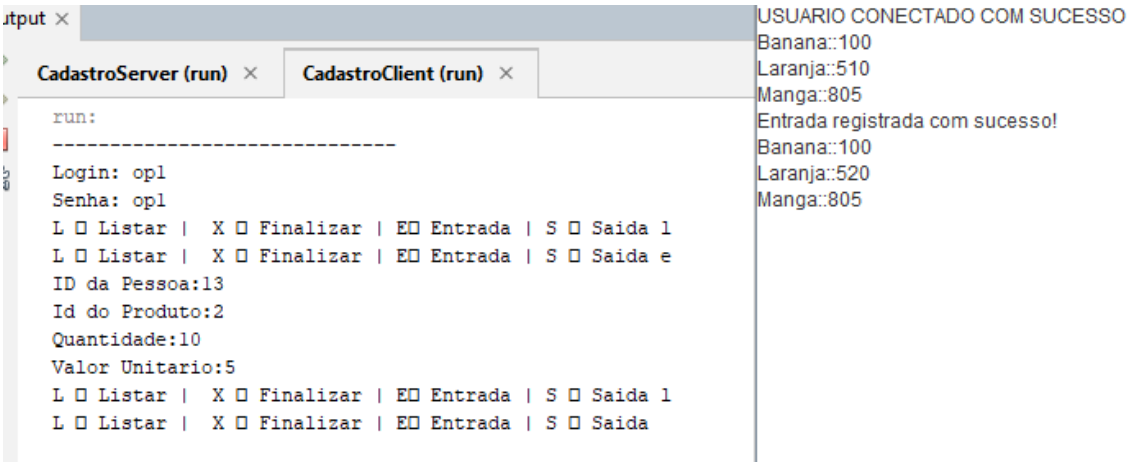
```
public class ThreadClient extends Thread {

    private ObjectInputStream in;
    private final JTextArea textArea;

    private JFrame frame;

    public ThreadClient(ObjectInputStream in) {
        this.in = in;
        frame = new JFrame("Mensagens do Servidor");
        textArea = new JTextArea(20, 50);
        textArea.setEditable(false);
        frame.add(new JScrollPane(textArea));
        frame.pack();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
    @Override
    public void run() {
        try {
            while (true) {
                Object data = in.readObject();
                String mensagem = (String) data;
                SwingUtilities.invokeLater(() -> {
                    textArea.append(mensagem + "\n");
                    textArea.setCaretPosition(textArea.getDocument().getLength()); // Rolagem
automática
                });
            }
        } catch (Exception e) {
            // Lidar com exceções, se necessário
```

```
    }
  }
}
```

Resultados

Funcionalidade Listar

Funcionalidade Entrada



Funcionalidade Saida

Funcionalidade Finalizar

```
Valor Unitario:5
L □ Listar |  X □ Finalizar | E□ Entrada | S □ Saida 1
L □ Listar |  X □ Finalizar | E□ Entrada | S □ Saida x
PROGRAMA FINALIZADO!BUILD SUCCESSFUL (total time: 5 minutes 34 seconds)
```

Análise

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

R: Em algumas linguagens e bibliotecas, você pode usar callbacks ou eventos assíncronos para lidar com respostas de servidores sem a necessidade explícita de criar threads. Por exemplo, em JavaScript, você pode fazer solicitações AJAX e definir funções de callback que serão executadas quando a resposta do servidor estiver pronta.

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

R: Executar uma determinada tarefa de maneira assíncrona no thread de despacho de eventos (Event Dispatch Thread - EDT), adicionando uma tarefa à fila de eventos da EDT para ser executada posteriormente. Esta é responsável por gerenciar todos os eventos de interface do usuário em aplicações Swing,

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

R: Os sockets em Java operam com fluxos de entrada (InputStream) e fluxos de saída (OutputStream), geralmente os objetos são serializados e desserializados.

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

R: Em operações síncronas, quando um cliente faz uma requisição através de um socket, o thread principal aguarda a resposta do servidor. Isso pode bloquear o thread principal até que a operação de leitura ou escrita nos sockets seja concluída.

Em operações assíncronas, quando uma operação de leitura ou escrita é iniciada em um socket, o thread principal continua sua execução sem esperar pelo resultado dessa operação. Isso permite que o thread principal execute outras tarefas enquanto aguarda a conclusão da operação assíncrona.

Ambas as abordagens, a concorrência é um fator importante a considerar. Com o comportamento síncrono, o uso excessivo de threads pode resultar em consumo excessivo de recursos do sistema. Com o comportamento assíncrono, é possível gerenciar melhor a concorrência, como usando um pool de threads para lidar com múltiplas operações assíncronas de maneira controlada.

Conclusão:

A escolha entre comportamento síncrono e assíncrono em clientes usando sockets Java depende das necessidades específicas da aplicação. Enquanto o síncrono oferece simplicidade de controle, o assíncrono proporciona melhor utilização de recursos e reatividade. Muitas vezes, uma combinação dessas abordagens pode ser a solução mais adequada, otimizando tanto o desempenho quanto a responsividade da aplicação.

Esta análise ressalta a importância de considerar os trade-offs entre simplicidade, eficiência e complexidade ao escolher a abordagem de comunicação em aplicações baseadas em sockets Java.