



Centro Universitário Estácio

Curso: Desenvolvimento Full Stack

Disciplina: Por Que Não Paralelizar?

Semestre: 3º

Aluno: Jonnatha Walben Saldanha da Silva

1º Procedimento - Criando o Servidor e Cliente de Teste

O objetivo desta prática é compreender e implementar um sistema de comunicação cliente-servidor utilizando sockets em Java. O servidor autentica usuários e fornece uma lista de produtos armazenados em um banco de dados, utilizando JPA (Java Persistence API) para a persistência dos dados. A prática abrange a configuração da conexão com o banco de dados, a implementação de sockets para comunicação entre cliente e servidor, e a manipulação de objetos serializáveis para a troca de dados.

Códigos Solicitados:

Aqui está um resumo dos códigos solicitados e apresentados ao longo do procedimento:

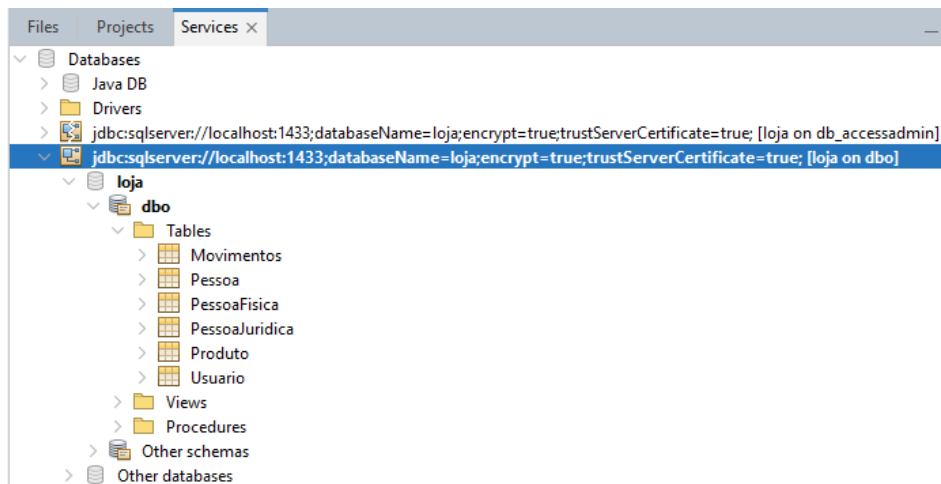
CadastroServer

```

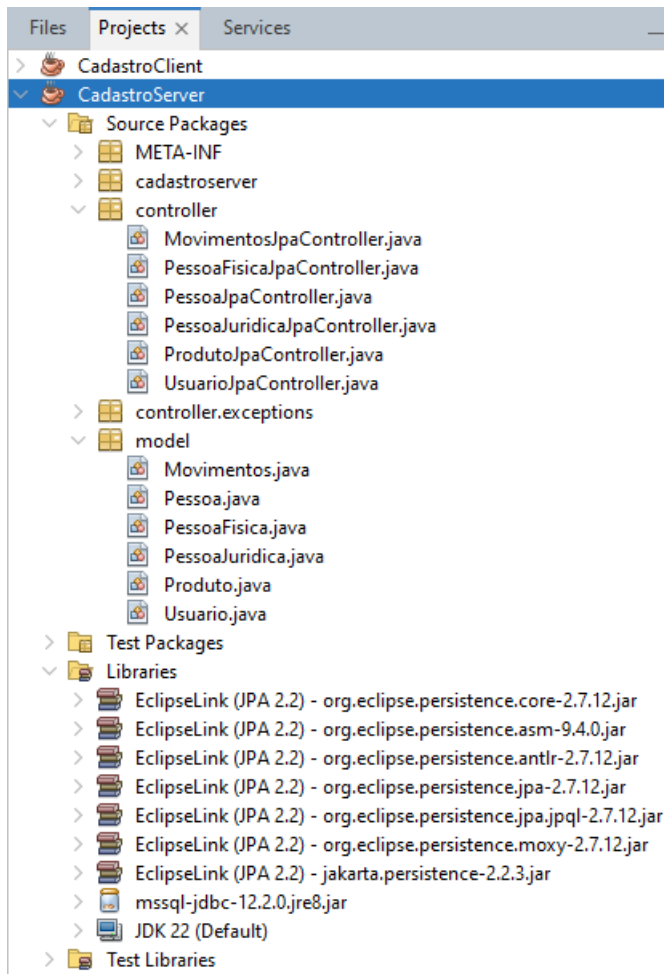
4
5 package cadastrserver;
6
7 /**
8  *
9  * @author jhon
10 */
11 import controller.ProdutoJpaController;
12 import controller.UsuarioJpaController;
13 import java.io.IOException;
14 import java.net.ServerSocket;
15 import javax.persistence.EntityManagerFactory;
16 import javax.persistence.Persistence;
17
18 public class CadastroServer {
19     public static void main(String[] args) {
20         EntityManagerFactory emf = Persistence.createEntityManagerFactory("CadastroServerPU");
21         ProdutoJpaController ctrlProduto = new ProdutoJpaController(emf);
22         UsuarioJpaController ctrlUsuario = new UsuarioJpaController(emf);
23
24         try (ServerSocket serverSocket = new ServerSocket(4321)) {
25             while (true) {
26                 // Aguardar conexão do cliente
27                 CadastroThread cadastroThread = new CadastroThread(ctrlProduto, ctrlUsuario, serverSocket.accept());
28
29                 // Iniciar uma nova thread para lidar com a comunicação
30                 Thread thread = new Thread(cadastroThread);
31                 thread.start();
32             }
33         } catch (IOException e) {
34             e.printStackTrace();
35         }
36     }
37 }

```

Conexão com o SQL Server



Pacote model / Pacote controller JPA / Biblioteca Eclipse / jar



Método `findUsuario`

```

30 public Usuario findUsuario(String nome, String senha) {
31     EntityManager em = getEntityManager();
32     try {
33         Query query = em.createQuery
34             (string: "SELECT u FROM Usuario u WHERE u.nome = :nome AND u.senha = :senha");
35         query.setParameter(string: "nome", o: nome);
36         query.setParameter(string: "senha", o: senha);
37         return (Usuario) query.getSingleResult();
38     } catch (NoResultException ex) {
39         System.out.println(x: "Nenhum usuário encontrado com as credenciais fornecidas.");
40         return null;
41     } finally {
42         em.close();
43     }
44 }

```

Ativar o Windows

CadastroThread

```

5 package cadastrserver;
6 /**
7  *
8  * @author jhon
9  */
10 import controller.ProdutoJpaController;
11 import controller.UsuarioJpaController;
12 import model.Produto;
13 import model.Usuario;
14 import java.io.IOException;
15 import java.io.ObjectInputStream;
16 import java.io.ObjectOutputStream;
17 import java.net.Socket;
18 import java.util.List;
19
20 public class CadastroThread implements Runnable {
21     private final ProdutoJpaController ctrl;
22     private final UsuarioJpaController ctrlUsu;
23     private final Socket s1;
24
25     public CadastroThread(ProdutoJpaController ctrl, UsuarioJpaController ctrlUsu, Socket s1) {
26         this.ctrl = ctrl;
27         this.ctrlUsu = ctrlUsu;
28         this.s1 = s1;
29     }
30
31     @Override
32     public void run() {
33         try {
34             // Inicializar canais de entrada e saída
35             ObjectOutputStream out = new ObjectOutputStream(s1.getOutputStream());
36             ObjectInputStream in = new ObjectInputStream(s1.getInputStream());
37
38             // Receber login e senha
39             String login = (String) in.readObject();
40             String senha = (String) in.readObject();
41
42             // Validar credenciais
43             Usuario usuario = ctrlUsu.findUsuario(login, senha);
44             if (usuario == null) {
45                 out.writeObject("Falha na autenticação");
46                 s1.close(); // Desconectar cliente se as credenciais forem inválidas
47                 return;
48             }
49
50             out.writeObject("Usuario conectado com sucesso.");
51
52             // Loop de resposta
53             while (true) {
54                 // Receber comando do cliente
55                 String comando = (String) in.readObject();
56
57                 // Responder ao comando "L" com o conjunto de produtos
58                 if (comando.equals("L")) {
59                     List<Produto> produtos = ctrl.findProdutoEntities();
60                     out.writeObject(produtos);
61                     out.flush();
62                 }
63
64                 // Implementar lógica para outros comandos, se necessário
65             }
66         } catch (IOException | ClassNotFoundException e) {
67             e.printStackTrace();
68         }
69     }
70 }

```

ProdutoJpaController

```

5 package controller;
6 /**
7  *
8  * @author jhon
9  */
10 import model.Produto;
11
12 import javax.persistence.EntityManager;
13 import javax.persistence.EntityManagerFactory;
14 import javax.persistence.TypedQuery;
15 import java.util.List;
16
17 public class ProdutoJpaController {
18     private EntityManagerFactory emf = null;
19
20     public ProdutoJpaController(EntityManagerFactory emf) {
21         this.emf = emf;
22     }
23
24     public EntityManager getEntityManager() {
25         return emf.createEntityManager();
26     }
27
28     public void create(Produto produto) {
29         EntityManager em = getEntityManager();
30         try {
31             em.getTransaction().begin();
32             em.persist(produto);
33             em.getTransaction().commit();
34         } finally {
35             em.close();
36         }
37     }
38
39     public void edit(Produto produto) {
40         EntityManager em = getEntityManager();
41         try {
42             em.getTransaction().begin();
43             produto = em.merge(produto);
44             em.getTransaction().commit();
45         } finally {
46             em.close();
47         }
48     }
49
50     public void destroy(Long id) {
51         EntityManager em = getEntityManager();
52         try {
53             em.getTransaction().begin();
54             Produto produto = em.find(Produto.class, id);
55             if (produto != null) {
56                 em.remove(produto);
57             }
58             em.getTransaction().commit();
59         } finally {
60             em.close();
61         }
62     }
63
64     public Produto findProduto(Long id) {
65         EntityManager em = getEntityManager();
66         try {
67             return em.find(Produto.class, id);
68         } finally {
69             em.close();
70         }
71     }
72 }

```

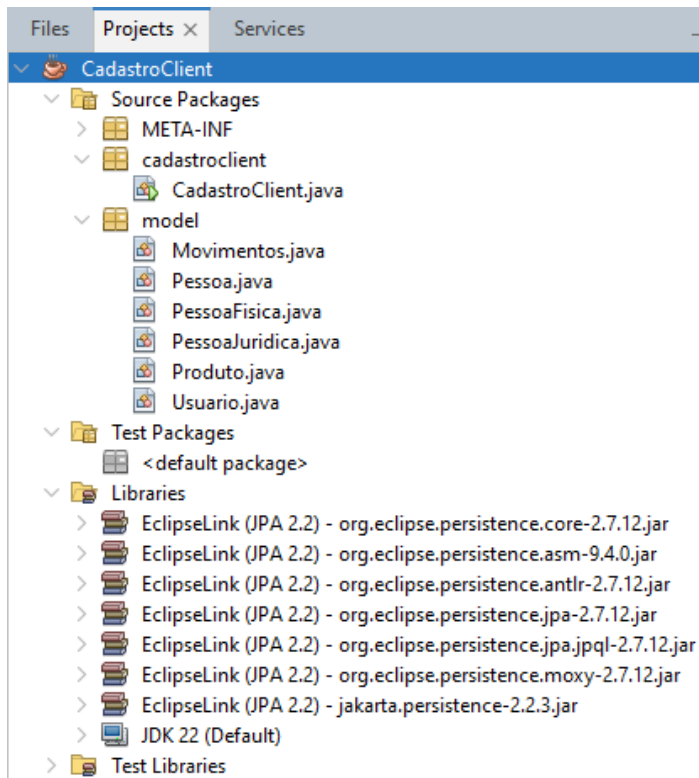
UsuarioJpaController

```

5 package controller;
6 /**
7  *
8  * @author jhon
9  */
10 import model.Usuario;
11 import javax.persistence.EntityManager;
12 import javax.persistence.EntityManagerFactory;
13 import javax.persistence.NoResultException;
14 import javax.persistence.Query;
15
16 public class UsuarioJpaController {
17     private EntityManagerFactory emf;
18
19     public UsuarioJpaController(EntityManagerFactory emf) {
20         this.emf = emf;
21     }
22
23     public EntityManager getEntityManager() {
24         return emf.createEntityManager();
25     }
26
27     public Usuario findUsuario(String nome, String senha) {
28         EntityManager em = getEntityManager();
29         try {
30             Query query = em.createQuery
31                 ("SELECT u FROM Usuario u WHERE u.nome = :nome AND u.senha = :senha");
32             query.setParameter("nome", nome);
33             query.setParameter("senha", senha);
34             return (Usuario) query.getSingleResult();
35         } catch (NoResultException ex) {
36             System.out.println("Nenhum usuário encontrado com as credenciais fornecidas.");
37             return null;
38         } finally {
39             em.close();
40         }
41     }
42 }
43

```

Criação do Cliente Teste



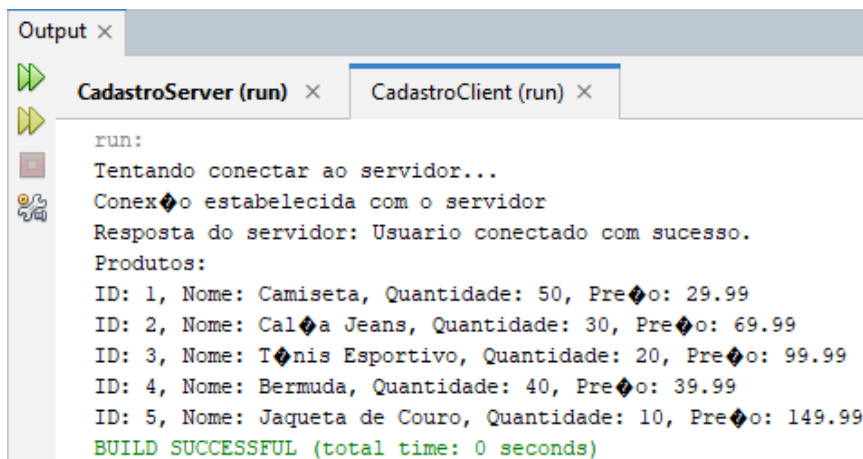
CadastroClient

```

5 package cadastroclient;
6 /**
7  *
8  * @author jhon
9  */
10 import model.Produto;
11
12 import java.io.IOException;
13 import java.io.ObjectInputStream;
14 import java.io.ObjectOutputStream;
15 import java.net.Socket;
16 import java.util.List;
17
18 public class CadastroClient {
19     public static void main(String[] args) {
20         try {
21             System.out.println("Tentando conectar ao servidor...");
22             Socket socket = new Socket("localhost", 4321);
23             System.out.println("Conexão estabelecida com o servidor");
24
25             ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
26             ObjectInputStream in = new ObjectInputStream(socket.getInputStream());
27
28             // Enviar login e senha
29             out.writeObject("aaa");
30             out.writeObject("aaa");
31
32             // Receber resposta do servidor
33             String response = (String) in.readObject();
34             System.out.println("Resposta do servidor: " + response);
35
36             if ("Usuario conectado com sucesso.".equals(response)) {
37                 // Enviar comando "L" para listar produtos
38                 out.writeObject("L");
39
40                 // Receber e apresentar lista de produtos
41                 Object obj = in.readObject();
42                 if (obj instanceof List) {
43                     List<Produto> produtos = (List<Produto>) obj;
44                     System.out.println("Produtos:");
45                     for (Produto produto : produtos) {
46                         System.out.println("ID: " + produto.getIDProduto() + ", Nome: " + produto.getNome() + ", Quantidade: " + produto.getQuantidade());
47                     }
48                 } else {
49                     System.out.println("Falha na autenticação");
50                 }
51             }
52
53             // Fechar conexões
54             in.close();
55             out.close();
56             socket.close();
57         } catch (IOException | ClassNotFoundException e) {
58             e.printStackTrace();
59         }
60     }
61 }
62

```

Resultado saída cliente



```
run:
Tentando conectar ao servidor...
Conexão estabelecida com o servidor
Resposta do servidor: Usuario conectado com sucesso.
Produtos:
ID: 1, Nome: Camiseta, Quantidade: 50, Preço: 29.99
ID: 2, Nome: Calça Jeans, Quantidade: 30, Preço: 69.99
ID: 3, Nome: Tênis Esportivo, Quantidade: 20, Preço: 99.99
ID: 4, Nome: Bermuda, Quantidade: 40, Preço: 39.99
ID: 5, Nome: Jaqueta de Couro, Quantidade: 10, Preço: 149.99
BUILD SUCCESSFUL (total time: 0 seconds)
```

Conclusão

1. Importância das portas para a conexão com servidores:

As portas são fundamentais para a comunicação entre clientes e servidores em uma rede. Cada aplicação que utiliza a rede precisa de uma porta única para enviar e receber dados. As portas garantem que os dados sejam direcionados para os aplicativos corretos em um servidor. Por exemplo, um servidor web geralmente escuta na porta 80 para solicitações HTTP e na porta 443 para solicitações HTTPS.

2. Classes de entrada e saída `ObjectInputStream` e `ObjectOutputStream`:

As classes `ObjectInputStream` e `ObjectOutputStream` são utilizadas para realizar entrada e saída de objetos em Java, respectivamente. Elas são usadas para serializar e desserializar objetos, permitindo que objetos complexos sejam convertidos em fluxos de bytes que podem ser transmitidos pela rede ou armazenados em arquivos. Os objetos transmitidos devem ser serializáveis para que possam ser convertidos em bytes e transferidos pela rede de forma eficiente e segura.

3. Garantia do isolamento do acesso ao banco de dados utilizando classes de entidades JPA no cliente:

Utilizando as classes de entidades JPA no cliente, como visto no exemplo, é possível garantir o isolamento do acesso ao banco de dados porque o acesso ao banco de dados é controlado pelo servidor. O cliente envia solicitações para o servidor, que então acessa o banco de dados e retorna os resultados. Isso ajuda

a manter uma arquitetura de sistema distribuído onde a lógica de negócios e o acesso aos dados são centralizados no servidor, enquanto o cliente lida com a interface do usuário e a interação com o usuário. Dessa forma, mesmo que o cliente possua classes de entidades JPA, ele não acessa diretamente o banco de dados, o que contribui para a segurança e a integridade dos dados.

2º Procedimento – Servidor Completo e Cliente Assíncrono

Objetivo da Prática

O objetivo da prática do 2º procedimento é implementar um servidor de cadastro de produtos que seja capaz de processar comandos de entrada (E) e saída (S) de produtos, gerenciando as movimentações através de um banco de dados. O cliente deve ser capaz de se conectar ao servidor, enviar comandos, receber respostas e atualizar uma interface gráfica de forma assíncrona, garantindo uma interação eficiente e responsiva com o sistema.

Códigos Solicitados

Segunda versão da Thread de comunicação

```
5 package cadastroclient;
6 /**
7  *
8  * @author jhon
9  */
10 import javax.swing.*;
11 import java.io.IOException;
12 import java.io.ObjectInputStream;
13 import java.util.List;
14 import model.Produto;
15
16 public class ThreadClient extends Thread {
17     private final ObjectInputStream entrada;
18     private final JTextArea textArea;
19
20     public ThreadClient(ObjectInputStream entrada, JTextArea textArea) {
21         this.entrada = entrada;
22         this.textArea = textArea;
23     }
24
25     @Override
26     public void run() {
27         try {
28             while (true) {
29                 Object obj = entrada.readObject();
30                 if (obj instanceof String string) {
31                     SwingUtilities.invokeLater(() -> textArea.append(string + "\n"));
32                 } else if (obj instanceof List<?>) {
33                     SwingUtilities.invokeLater(() -> {
34                         List<Produto> produtos = (List<Produto>) obj;
35                         for (Produto produto : produtos) {
36                             textArea.append(produto.getNome() + " - " + produto.getQuantidade() + "\n");
37                         }
38                     });
39                 }
40             }
41         } catch (IOException | ClassNotFoundException e) {
42             //
43         }
44     }
```


Cliente assíncrono

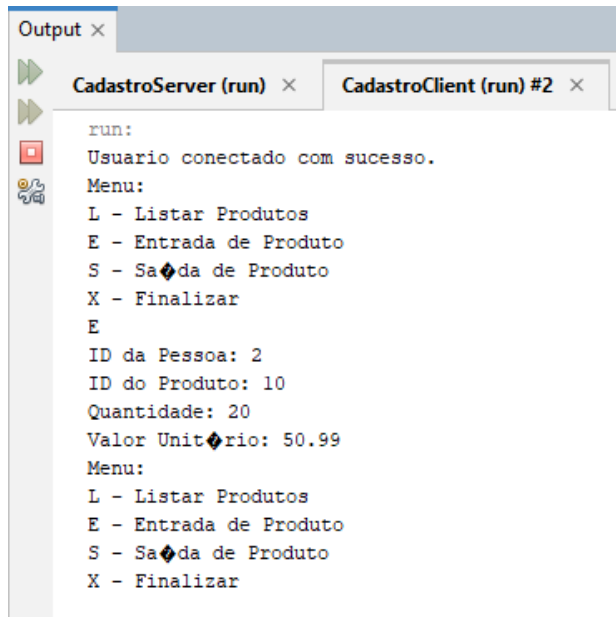
```
5 package cadastroclient;
6
7 import java.io.BufferedReader;
8 import java.io.IOException;
9 import java.io.InputStreamReader;
10 import java.io.ObjectInputStream;
11 import java.io.ObjectOutputStream;
12 import java.math.BigDecimal;
13 import java.net.Socket;
14
15 public class CadastroClientV2 {
16     public static void main(String[] args) {
17         try (Socket socket = new Socket("localhost", 4321);
18             ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
19             ObjectInputStream in = new ObjectInputStream(socket.getInputStream());
20             BufferedReader reader = new BufferedReader(new InputStreamReader(System.in))) {
21
22             // Enviar login e senha
23             out.writeObject("aaa");
24             out.writeObject("aaa");
25
26             // Receber mensagem de conexão
27             String message = (String) in.readObject();
28             System.out.println(message);
29
30             // Inicializar janela de saída
31             SaidaFrame saidaFrame = new SaidaFrame();
32             saidaFrame.setVisible(true);
33
34             // Inicializar thread de preenchimento assíncrono
35             ThreadClient threadClient = new ThreadClient(in, saidaFrame.texto);
36             threadClient.start();
37
38             while (true) {
39                 System.out.println("Menu:");
40                 System.out.println("L - Listar Produtos");
41                 System.out.println("E - Entrada de Produto");
42                 System.out.println("S - Saída de Produto");
43                 System.out.println("X - Finalizar");
44
45                 String command = reader.readLine();
46                 out.writeObject(command);
47
48                 if (command.equals("L")) {
49                     // A thread assíncrona cuidará da apresentação dos produtos
50                 } else if (command.equals("E") || command.equals("S")) {
51                     System.out.print("ID da Pessoa: ");
52                     int idPessoa = Integer.parseInt(reader.readLine());
53                     System.out.print("ID do Produto: ");
54                     int idProduto = Integer.parseInt(reader.readLine());
55                     System.out.print("Quantidade: ");
56                     int quantidade = Integer.parseInt(reader.readLine());
57                     System.out.print("Valor Unitário: ");
58                     String valorUnitarioStr = reader.readLine().replace(".", "");
59                     BigDecimal valorUnitario = new BigDecimal(valorUnitarioStr);
60
61                     // Enviar dados do movimento
62                     out.writeObject(idPessoa);
63                     out.writeObject(idProduto);
64                     out.writeObject(quantidade);
65                     out.writeObject(valorUnitario);
66                 } else if (command.equals("X")) {
67                     break;
68                 }
69             }
70         } catch (IOException | ClassNotFoundException e) {
71             e.printStackTrace();
72         }
73     }
74 }
75
```

13/06/2024, 23:10 CadastroClientV2.java

SaidaFrame

```
5 package cadastroclient;
6
7 /**
8  *
9  * @author jhon
10 */
11 import javax.swing.*;
12
13 public class SaidaFrame extends JDialog {
14     public JTextArea texto;
15
16     public SaidaFrame() {
17         // Definir dimensões da janela
18         setBounds(100, 100, 400, 300);
19         // Definir o status modal como false
20         setModal(false);
21         // Inicializar o JTextArea
22         texto = new JTextArea();
23         // Adicionar o JTextArea à janela
24         add(new JScrollPane(texto));
25     }
26 }
```

Resultado saída cliente



```
run:
Usuario conectado com sucesso.
Menu:
L - Listar Produtos
E - Entrada de Produto
S - Saída de Produto
X - Finalizar
E
ID da Pessoa: 2
ID do Produto: 10
Quantidade: 20
Valor Unitário: 50.99
Menu:
L - Listar Produtos
E - Entrada de Produto
S - Saída de Produto
X - Finalizar
```

Análise

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

As Threads são utilizadas para executar tarefas em paralelo, permitindo que um programa continue a realizar outras operações enquanto aguarda uma resposta do servidor. No contexto do cliente, uma Thread separada (ThreadClient) pode ser criada para lidar com a leitura contínua de respostas do servidor, atualizando a interface do usuário conforme novos dados chegam, sem bloquear a interação do usuário com o sistema.

Para que serve o método invokeLater, da classe SwingUtilities?

O método invokeLater da classe SwingUtilities é usado para garantir que as atualizações na interface gráfica do usuário (GUI) sejam realizadas na Event Dispatch Thread (EDT), que é a Thread responsável pelo gerenciamento de eventos e atualização da interface no Swing. Isso é importante para evitar problemas de concorrência e garantir que a interface seja atualizada de forma segura e consistente.

Como os objetos são enviados e recebidos pelo Socket Java?

No Java, objetos são enviados e recebidos através de ObjectOutputStream e ObjectInputStream. Quando um objeto é escrito no ObjectOutputStream, ele é convertido em um fluxo de bytes que pode ser transmitido pela rede. Do lado receptor,

o `ObjectInputStream` reconstrói o objeto a partir do fluxo de bytes recebido. Para que um objeto possa ser transmitido dessa forma, ele deve implementar a interface `Serializable`.

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

- **Comportamento Síncrono:**

- No comportamento síncrono, as operações de leitura e escrita no socket são bloqueantes, ou seja, o programa para e espera a conclusão da operação antes de continuar. Isso pode simplificar o design do programa, mas pode levar a uma interface não responsiva se a operação demorar muito tempo.
- Vantagens:
 - Simplicidade no código, sem necessidade de gerenciar múltiplas Threads.
 - Fácil de entender e seguir o fluxo lógico do programa.
- Desvantagens:
 - Pode levar a uma interface não responsiva.
 - Bloqueio do processamento enquanto aguarda resposta.

- **Comportamento Assíncrono:**

- No comportamento assíncrono, as operações de leitura e escrita no socket são realizadas em Threads separadas, permitindo que o programa continue a executar outras tarefas enquanto aguarda a conclusão das operações de E/S. Isso melhora a responsividade do programa, especialmente em interfaces gráficas.
- Vantagens:
 - Interface mais responsiva, pois a Thread principal não é bloqueada.
 - Melhor desempenho em sistemas com muitas operações de E/S.
- Desvantagens:
 - Complexidade adicional no código devido ao gerenciamento de Threads.
 - Necessidade de sincronização para acessar recursos compartilhados.

Conclusão

O uso de Threads para tratamento assíncrono em aplicativos cliente-servidor em Java permite a criação de aplicações mais responsivas e eficientes. No caso do `CadastroClientV2`, a implementação de uma Thread separada para leitura de respostas do servidor, juntamente com o uso de `invokeLater` para atualizações seguras da GUI, demonstra como técnicas assíncronas podem melhorar a experiência do usuário sem complicar demasiadamente o código. Comparando com o comportamento síncrono, a abordagem assíncrona evita bloqueios indesejados e permite que a interface gráfica permaneça ativa e responsiva, mesmo durante operações de rede prolongadas.