Missão Prática | Nível 5 | Mundo 3

Lorena Rosa Borges Sanches - Matrícula: 202204376067

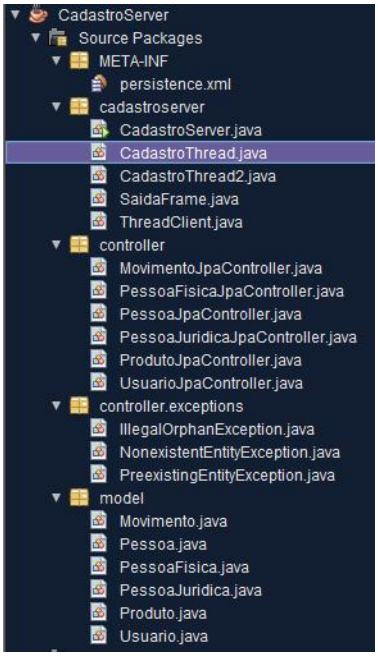
Estácio Campus Uberlândia/MG
Por que não paralelizar - 2023.2 – 3° Semestre Letivo
Desenvolvimento Full Stack
Link do Git: github.com/LorenaBorgesSanches/Mundo3-Pratica5

Objetivo da Prática

O propósito do trabalho a seguir é acompanhar a atividade prática elaborada como requisito da disciplina “RPG0018”.

Criando o Servidor

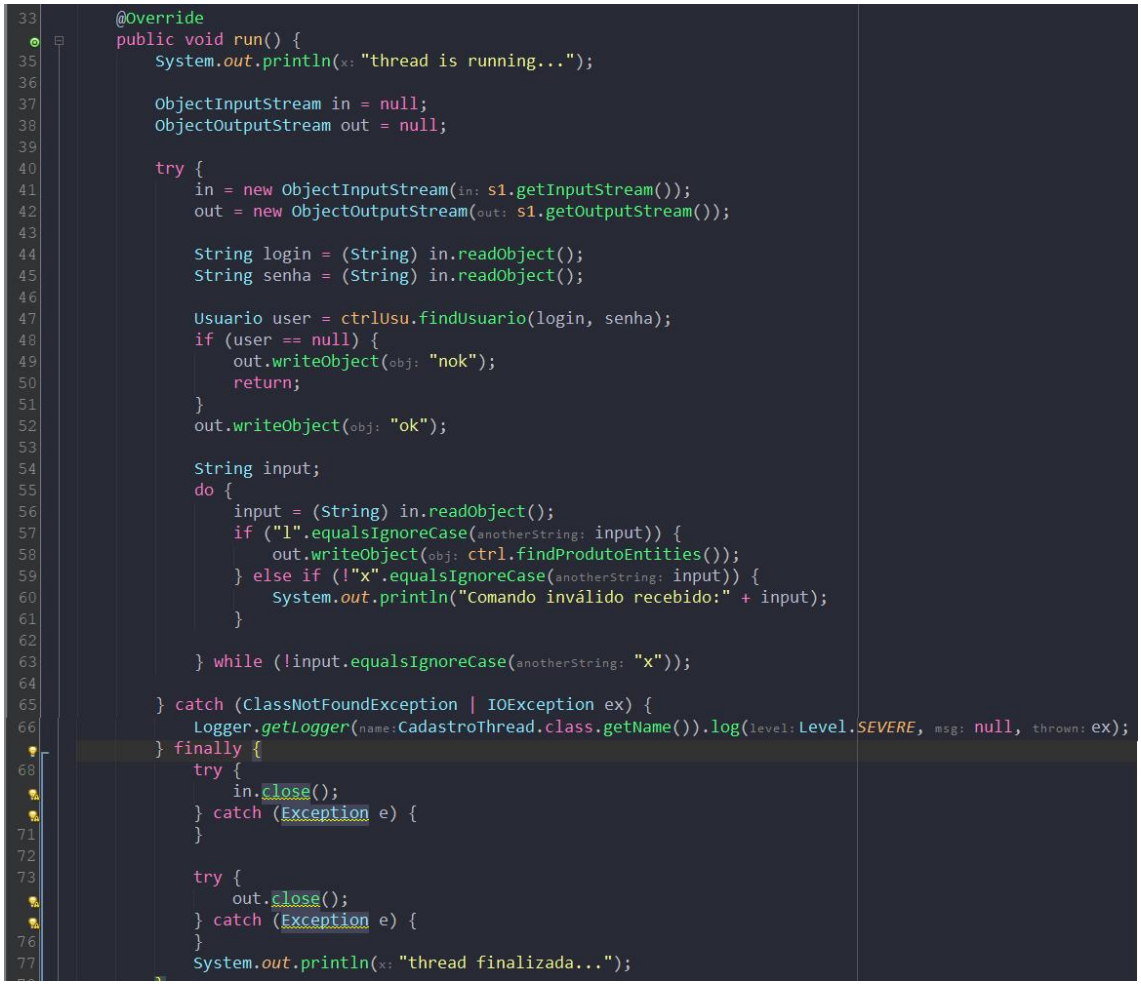
Estrutura:



Obs: Não estou colocando os prints das classes auto-geradas pelo NetBeans (model, controllers e controller.execptions).

Thread v1:

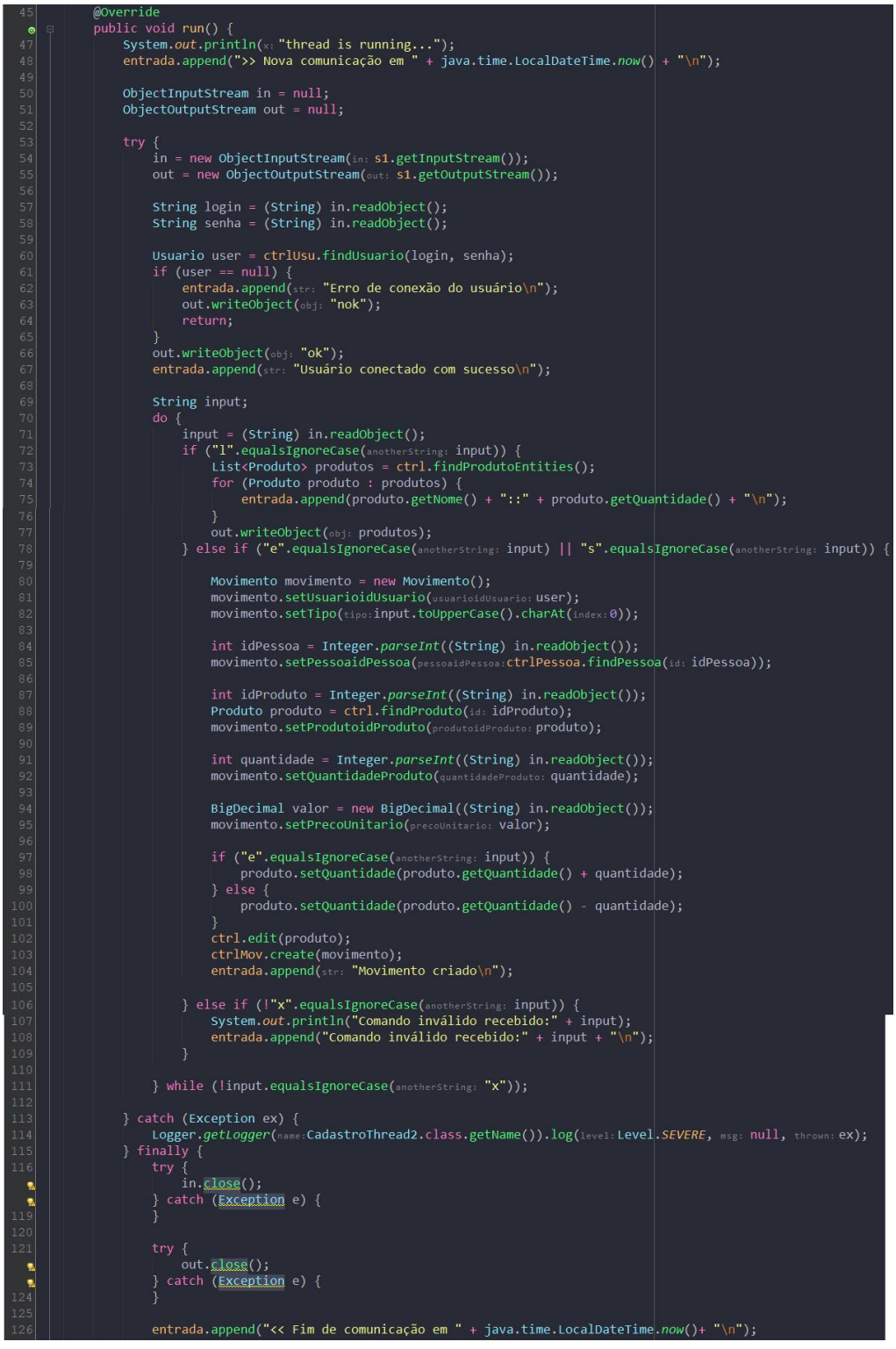
Captura de login e senha, validação do usuário e apenas a funcionalidade de listagem de produto.



```
33      @Override
34      public void run() {
35          System.out.println(x: "thread is running...");
36
37          ObjectInputStream in = null;
38          ObjectOutputStream out = null;
39
40          try {
41              in = new ObjectInputStream(in: s1.getInputStream());
42              out = new ObjectOutputStream(out: s1.getOutputStream());
43
44              String login = (String) in.readObject();
45              String senha = (String) in.readObject();
46
47              Usuario user = ctrlUsu.findUsuario(login, senha);
48              if (user == null) {
49                  out.writeObject(obj: "nok");
50                  return;
51              }
52              out.writeObject(obj: "ok");
53
54              String input;
55              do {
56                  input = (String) in.readObject();
57                  if ("l".equalsIgnoreCase(anotherString: input)) {
58                      out.writeObject(obj: ctrl.findProdutoEntities());
59                  } else if (!"x".equalsIgnoreCase(anotherString: input)) {
60                      System.out.println("Comando inválido recebido:" + input);
61                  }
62
63              } while (!input.equalsIgnoreCase(anotherString: "x"));
64
65          } catch (ClassNotFoundException | IOException ex) {
66              Logger.getLogger(name:CadastroThread.class.getName()).log(level:Level.SEVERE, msg: null, thrown:ex);
67          } finally {
68              try {
69                  in.close();
70              } catch (Exception e) {
71              }
72
73              try {
74                  out.close();
75              } catch (Exception e) {
76              }
77              System.out.println(x: "thread finalizada...");
```

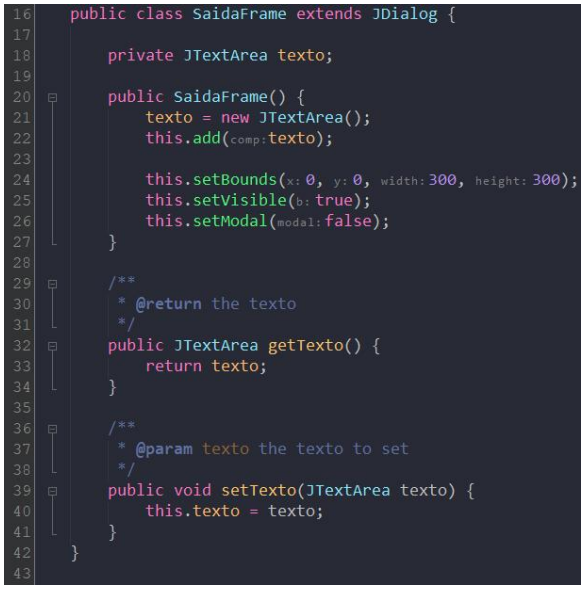
Thread v2:

Captura de login e senha, validação do usuário, funcionalidade de listar produtos e inclusão de movimentos. E utilização da variavel ‘entrada’ para retorno visual das operações em JDialog.



```
@Override
public void run() {
    System.out.println(x: "thread is running...");
    entrada.append(">> Nova comunicação em " + java.time.LocalDateTime.now() + "\n");

    ObjectInputStream in = null;
    ObjectOutputStream out = null;

    try {
        in = new ObjectInputStream(in: s1.getInputStream());
        out = new ObjectOutputStream(out: s1.getOutputStream());

        String login = (String) in.readObject();
        String senha = (String) in.readObject();

        Usuario user = ctrlUsu.findUsuario(login, senha);
        if (user == null) {
            entrada.append(str: "Erro de conexão do usuário\n");
            out.writeObject(obj: "nok");
            return;
        }
        out.writeObject(obj: "ok");
        entrada.append(str: "Usuário conectado com sucesso\n");

        String input;
        do {
            input = (String) in.readObject();
            if ("l".equalsIgnoreCase(anotherString: input)) {
                List<Produto> produtos = ctrl.findProdutoEntities();
                for (Produto produto : produtos) {
                    entrada.append(produto.getNome() + "::" + produto.getQuantidade() + "\n");
                }
                out.writeObject(obj: produtos);
            } else if ("e".equalsIgnoreCase(anotherString: input) || "s".equalsIgnoreCase(anotherString: input)) {

                Movimento movimento = new Movimento();
                movimento.setUsuarioidUsuario(usuarioidUsuario: user);
                movimento.setTipo(tipo:input.toUpperCase().charAt(index:0));

                int idPessoa = Integer.parseInt((String) in.readObject());
                movimento.setPessoaidPessoa(pessoaidPessoa:ctrlPessoa.findPessoa(id: idPessoa));

                int idProduto = Integer.parseInt((String) in.readObject());
                Produto produto = ctrl.findProduto(id: idProduto);
                movimento.setProdutoidProduto(produtoidProduto: produto);

                int quantidade = Integer.parseInt((String) in.readObject());
                movimento.setQuantidadeProduto(quantidadeProduto: quantidade);

                BigDecimal valor = new BigDecimal((String) in.readObject());
                movimento.setPrecoUnitario(precoUnitario: valor);

                if ("e".equalsIgnoreCase(anotherString: input)) {
                    produto.setQuantidade(produto.getQuantidade() + quantidade);
                } else {
                    produto.setQuantidade(produto.getQuantidade() - quantidade);
                }
                ctrl.edit(produto);
                ctrlMov.create(movimento);
                entrada.append(str: "Movimento criado\n");

            } else if (!"x".equalsIgnoreCase(anotherString: input)) {
                System.out.println("Comando inválido recebido:" + input);
                entrada.append("Comando inválido recebido:" + input + "\n");
            }

        } while (!input.equalsIgnoreCase(anotherString: "x"));

    } catch (Exception ex) {
        Logger.getLogger(name:CadastroThread2.class.getName()).log(level:Level.SEVERE, msg: null, thrown: ex);
    } finally {
        try {
            in.close();
        } catch (Exception e) {
        }

        try {
            out.close();
        } catch (Exception e) {
        }

        entrada.append("<< Fim de comunicação em " + java.time.LocalDateTime.now()+ "\n");
```

JDialog para exibir os resultados das requisições:



```java
public class SaidaFrame extends JDialog {

    private JTextArea texto;

    public SaidaFrame() {
        texto = new JTextArea();
        this.add(comp:texto);

        this.setBounds(x: 0, y: 0, width: 300, height: 300);
        this.setVisible(b: true);
        this.setModal(modal: false);
    }

    /**
     * @return the texto
     */
    public JTextArea getTexto() {
        return texto;
    }

    /**
     * @param texto the texto to set
     */
    public void setTexto(JTextArea texto) {
        this.texto = texto;
    }
}
```

Thread Cliente:

Para iniciar as novas conexões recebidas por ‘clientes’ e iniciar a Thread2 que trata as requisições.

```java
public class ThreadClient extends Thread {

    public final JTextArea entrada;

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

    @Override
    public void run() {
        try {
            EntityManagerFactory em = Persistence.createEntityManagerFactory(persistenceUnitName:"CadastroServerPU");
            ProdutoJpaController ctrl = new ProdutoJpaController(emf: em);
            UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf: em);
            PessoaJpaController ctrlPessoa = new PessoaJpaController(emf: em);
            MovimentoJpaController ctrlMov = new MovimentoJpaController(emf: em);

            ServerSocket serverSocket = new ServerSocket(port:4321);

            while (true) {
                Socket s1 = serverSocket.accept();
                CadastroThread2 cadastroThread = new CadastroThread2(ctrl, ctrlUsu, ctrlPessoa, ctrlMov, entrada, s1);
                cadastroThread.start();
            }
        } catch (Exception ex) {
            Logger.getLogger(name:ThreadClient.class.getName()).log(level:Level.SEVERE, msg: null, thrown: ex);
        }
    }
}
```

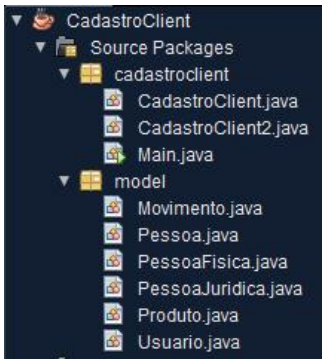
CadastroServer:

Ponto de inicio do programa, inicializa o JDialog e ThreadClient que fica escutando por novas conexões. Utilizando o invokeLater, como solicitado.

```java
    public static void main(String[] args) throws IOException {
        SwingUtilities.invokeLater(() -> {
            SaidaFrame frame = new SaidaFrame();
            ThreadClient client = new ThreadClient(entrada: frame.getTexto());
            client.start();
        });
    }
```

Criando o Cliente

Estrutura:



Obs: Não estou colocando os prints das classes auto-geradas pelo NetBeans (model).

CadastroClient:

Envio FIXO de usuário e senha como op1, tratativa do login, faz apenas uma solicitação de listagem(L) de produtos, printa, e depois finalização a conexão com o server (X).

```
25      public void Execute() throws IOException, ClassNotFoundException {
26          Socket clientSocket = null;
27          ObjectInputStream in = null;
28          ObjectOutputStream out = null;
29          try {
30              clientSocket = new Socket(address: InetAddress.getByName(host:"localhost"), port:4321);
31              out = new ObjectOutputStream(out: clientSocket.getOutputStream());
32              in = new ObjectInputStream(in: clientSocket.getInputStream());
33
34              out.writeObject(obj: "op1");
35              out.writeObject(obj: "op1");
36
37              String result = (String) in.readObject();
38              if (!"ok".equals(anObject: result)) {
39                  System.out.println(x: "Erro de login");
40                  return;
41              }
42              System.out.println(x: "Login com sucesso");
43
44              out.writeObject(obj: "L");
45
46              List<Produto> Produtos = (List<Produto>) in.readObject();
47              for (Produto produto : Produtos) {
48                  System.out.println(x: produto.getNome());
49              }
50
51              out.writeObject(obj: "X");
52
53          } finally {
54              if (out != null) {
55                  out.close();
56              }
57              if (in != null) {
58                  in.close();
59              }
60              if (clientSocket != null) {
61                  clientSocket.close();
62              }
63          }
64      }
65  }
66
```

CadastroCliente2:

Envio de usuário e senha digitado pelo usuário, tratativa do login, recebe o comando do usuário e trata cada um deles (L – Listar, E – Entrada, S – Saída, X – Finalizar).

```
    public void Execute() throws IOException, ClassNotFoundException {
        Socket clientSocket = null;
        ObjectInputStream in = null;
        ObjectOutputStream out = null;
        BufferedReader reader = new BufferedReader(new InputStreamReader(in: System.in));
        try {
            clientSocket = new Socket(address: InetAddress.getByName(host:"localhost"), port:4321);
            out = new ObjectOutputStream(out: clientSocket.getOutputStream());
            in = new ObjectInputStream(in: clientSocket.getInputStream());

            System.out.println(x: "Digite o Usuário: ");
            out.writeObject(obj: reader.readLine());

            System.out.println(x: "Digite a Senha: ");
            out.writeObject(obj: reader.readLine());

            String result = (String) in.readObject();
            if (!"ok".equals(anObject: result)) {
                System.out.println(x: "Erro de login");
                return;
            }
            System.out.println(x: "Login com sucesso");

            String comando;
            do {
                System.out.println(x: "Digite o Comando (L – Listar, E – Entrada, S – Saída, X – Finalizar): ");
                comando = reader.readLine();
                out.writeObject(obj: comando);

                if ("l".equalsIgnoreCase(anotherString: comando)) {

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

                    for (Produto produto : Produtos) {
                        System.out.println(x: produto.getNome());
                    }
                } else if ("e".equalsIgnoreCase(anotherString: comando) || "s".equalsIgnoreCase(anotherString: comando)) {
                    System.out.println(x: "Digite o id da Pessoa");
                    String idPessoa = reader.readLine();

                    System.out.println(x: "Digite o id do Produto");
                    String idProduto = reader.readLine();

                    System.out.println(x: "Digite a quantidade do Produto");
                    String quantidade = reader.readLine();

                    System.out.println(x: "Digite o valor do Produto");
                    String valor = reader.readLine();

                    out.writeObject(obj: idPessoa);
                    out.writeObject(obj: idProduto);
                    out.writeObject(obj: quantidade);
                    out.writeObject(obj: valor);
                }

            } while (!"x".equalsIgnoreCase(anotherString: comando));

        } finally {
            if (out != null) {
                out.close();
            }
            if (in != null) {
                in.close();
            }
            if (clientSocket != null) {
```

Main:

Apenas inicia um CadastroClient2:

```
    public static void main(String[] args) throws IOException, ClassNotFoundException {
        CadastroClient2 cliente = new CadastroClient2();
        cliente.Execute();
    }
```

Análise e conclusão:

1.1. Como funcionam as classes Socket e ServerSocket?

As classes Socket e ServerSocket são utilizadas para possibilitar a comunicação entre o cliente e servidor. Cada uma delas desempenha um papel nessa relação. O ServerSocket é a classe que é utilizado no lado do servidor, ele aguarda as conexões dos clientes. Quando a classe ServerSocket estabelece a conexão em uma porta com um cliente é criado um objeto Socket. O Socket é a parte do cliente, que se conectará ao servidor e deverá passar o endereço IP da porta do servidor que deseja se conectar e ao final deve ser finalizado pelo método close( ).

1.2. Qual a importância das portas para conexão com servidores?

As portas possuem grande relevância nas conexões aos servidores. Dentre as mais importantes podemos citar sobre a identificação dos serviços disponibilizados em cada porta, por exemplo a porta 80 é associada ao servidor HTTP, a porta 587 é mais comum para emails, etc. Além disso, há o controle de segurança através de firewalls e os controles de tráfegos. Enfim, a portas são de grande importância na organização, direcionamento e na segurança das conexões aos servidores.

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

A classe ObjectInputStream serve para persistir os objetos enquanto a classe ObjectOutputStream servem para ler os objetos que foi desserializado. Os objetos devem ser serializáveis, porque é através da interface Serializable que o arquivo pode ser convertido em bytes e depois reconstruido de forma que nenhum dado é perdido.

1.4. Por que, mesmo utilizando as classes de entidade JPA no cliente, foi possível garantir o isolamento do acesso ao banco de dados?

Porque o projeto cliente apenas tem acesso ao contrato do modelo do banco de dados através das classes do JPA (ele sabe o que é um produto, uma pessoa, um movimento, etc) mas não tem acesso direto ao banco de dados, pois o conexão de fato com o banco de dados foi aberta exclusivamente no projeto de Server.

2º Procedimento – Servidor Completo e Cliente Assíncrono

Análise e conclusão?

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

As threads podem tratar as respostas do servidor em threads separadas, permitindo a otimização do processamento. Quando a aplicação recebe uma solicitação/resposta, ela vai ser tratada por uma thread, permitindo que a thread principal continue a processar outras demandas. Isso permite que a aplicação processe mais requisições em menos tempo.

2.2. Para que Serve o método invokeLater, da classe SwingUtilities?

O método invokeLater da classe SwingUtilities(o Swing é um framework responsável pelo gerenciamento de interface) é utilizado para enfileirar as demandas a fila de threads e executa-las assim que possível.

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

Os objetos são enviados e recebidos pelo Socket Java através das classes ObjectOutputStream e ObjectInputStream (e também do ServerSocket e o Socket para fazer a comunicação entre cliente e servidor) e a função dessas classes é a de persistir os objetos e recupera-los. Deve ser implementada a interface Serializable na classe para que os objetos possam ser convertidos em bytes(serializados), e posteriormente, serem reconstituídos(desserializados).

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

No comportamento assíncrono as threads não ficam inativas em operações que demandam tempo, pois essas são tratadas sem que bloqueie a thread principal. Quando há muitas tarefas simultâneas é interessante adotar o método assíncrono para melhor desempenho do sistema, pois dessa forma terá uso eficiente dos recursos. No entanto, aumenta o grau de complexidade, pois deverá haver tratamento de threads que precisam da execução de outra para atualizar objetos e controle para não haver alterações do mesmo objeto ao mesmo tempo, evitando a inconsistência.

Na utilização de comportamento síncrono cada requisição bloqueia a thread até que finalize a operação. Com isso, há o benefício de ser mais simples de compreender e debugar, porque as tarefas serão executadas uma por vez. Além disso, não será necessário o tratamento de sobrecarga e o gerenciamento dos recursos. Porém, quando há apenas uma thread para lidar com varias demandas, pode ocorrer o bloqueio do processamento, que é quando a thread fica inativa esperando uma demanda finalizar. O impacto de um bloqueio do processamento é negativo,uma vez que compromete o desempenho do programa, tornando-o lento, pois aquela thread inativa não vai processar outra tarefa enquanto aguarda a finalização daquela, portanto, a cpu fica inutilizada, gerando mau uso de recurso e possível atraso em outras tarefas que dependia daquela thread inativa.