Vamos integrar sistemas

Walber do Carmo Farias, 202303752547

Polo de Abaetetuba (PA)
Nivel 5 – Por que não paralelizar – 2023.1 – 2024.1

Objetivo da Prática

Os objetivos da prática foi criar servidores e clientes Java baseados em Sockets, tanto síncronos quanto assíncronos, utilizando Threads para processos paralelos. Para ao final ter desenvolvido um servidor Java com acesso a banco de dados via JPA e implementado clientes síncronos e assíncronos, usando Threads para suportar múltiplos clientes e respostas assíncronas.

1º Procedimento | Criando o Servidor e Cliente de Teste

Inserir neste campo, **de forma organizada**, todos os códigos do roteiro do 1º Procedimento da Atividade Prática, os resultados da execução do código e a Análise e Conclusão:

- Criando o método findUsuario na classe UsuarioOperadorJpaController para que ele seja responsável de buscar o usuário pelo login e senha, no banco de dados, por meio de uma consulta JPA.

```
public UsuarioOperador findUsuario(String login, String senha){
    EntityManager em = getEntityManager();
    try {
        TypedQuery<UsuarioOperador> query = em.createQuery(
                "SELECT u FROM UsuarioOperador u WHERE u.nome = :nome AND u.senha = :senha", UsuarioOperador.class);
       query.setParameter("nome", login);
       query.setParameter("senha", senha);

       try {
           return query.getSingleResult();
       } catch (NoResultException e) {
           return null;
       }
   } finally {
       em.close();
   }

}

private EntityManager getEntityManager() {
    return emf.createEntityManager();
}
```

- Criando a classe CadastroThread que herda do objeto Thread. Os atributos ctrl, crtlUsu e s1, para armazenam uma instância do ProdutoJpaController, do UsuarioOperadorJpaController e do Socket, respectivamente. Criação do método construtor que recebe como parâmetro as instâncias e armazena elas nos seus respectivos atributos.

```
public class CadastroThread extends Thread {

    private final ProdutoJpaController ctrl;
    private final UsuarioOperadorJpaController ctrlUsu;
    private final Socket s1;

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

- Implementando o método run, com os canais de entrada e saída do socket para que seja possível obter o login e a senha por meio do método readObject() da entrada do socket, verificação das credenciais pelo método findUsuario. Na

sequência, com o usuário logado, ele entra em um loop de resposta, o qual, a utilizar o comando “L” na entrada, o sistema retorna uma lista com os produtos cadastrados.

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

        UsuarioOperador usuario = ctrlUsu.findUsuario(login, senha);

        if (usuario == null) {
            out.writeObject("Credenciais inválidas. Conexão encerrada.");
            s1.close();
            return;
        }

        out.writeObject("Usuário logado com sucesso");

        while (true) {
            String command = (String) in.readObject();

            if ("L".equals(command)) {
                List<Produto> produtos = ctrl.findProdutos();
                out.writeObject(produtos);
            } else {
                out.writeObject("Comando desconhecido");
            }
        }

    } catch (IOException | ClassNotFoundException e) {
        System.err.println("Erro na comunicação com o cliente: " + e.getMessage());
    }
}
```

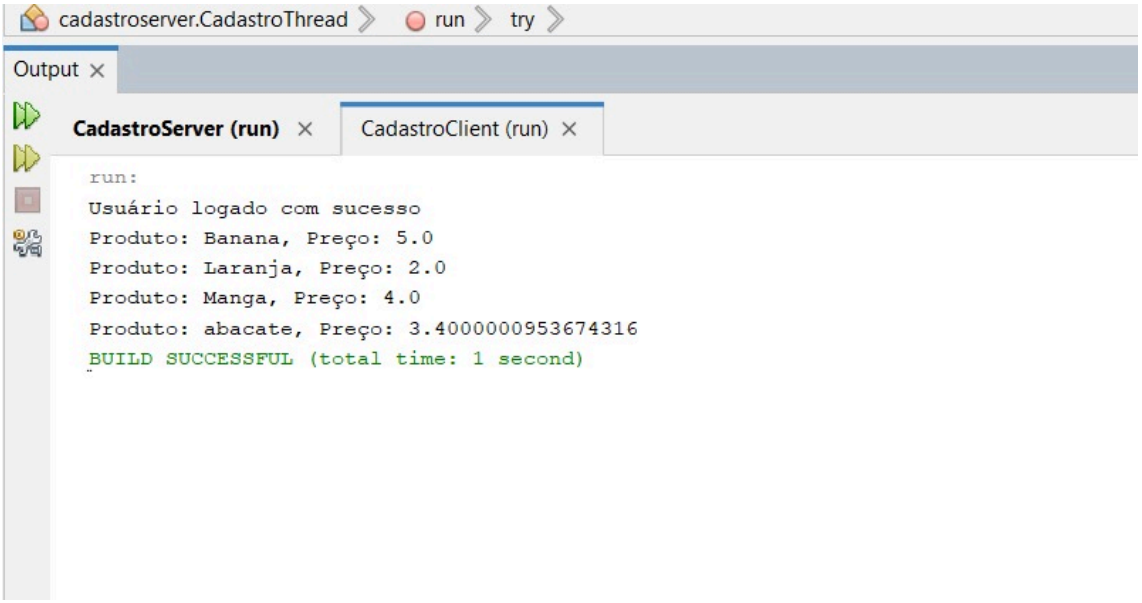
- Implementando o método main, com a instância de um objeto EntityManagerFactory, um objeto ProdutoJpaController e um objeto UsuarioJpaController, além do ServerSocket escutando a porta 4321 e um laço de repetição infinito para ouvir a porta do Socket quando uma requisição do cliente for feita.

```java
public class Main {
    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("CadastroServerPU");
        ProdutoJpaController produtoController = new ProdutoJpaController(emf);
        UsuarioOperadorJpaController usuarioController = new UsuarioOperadorJpaController(emf);

        try (ServerSocket serverSocket = new ServerSocket(4321)) {
            System.out.println("Servidor iniciado na porta 4321");

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

                CadastroThread thread = new CadastroThread(produtoController, usuarioController, clientSocket);
                thread.start();
            }
        } catch (IOException e) {
            System.err.println("Erro ao iniciar o servidor: " + e.getMessage());
        } finally {
            emf.close();
        }
    }
}
```

- Criação do cliente de teste CadastroCliente, o qual contém uma instância de um socket apontando para a porta 4321, com os objetos de entrada e saída do socket. Após, a chamada do método writeObject, da saída, para que o servidor consiga ler. Se o login for bem efetuado, o servidor devolve uma resposta de “Usuário logado com sucesso” para a entrada do Socket do cliente, que foi lido com o método readObject, da entrada do Socket. A mensagem foi verificada por meio do método equals para concluir se o usuário foi logado. Após isso a saída do Socket do cliente envia “L” para o servidor, que deve devolver uma lista de produtos, sendo utilizado o método readObject novamente, para ler a entrada do Socket do cliente. A lista de produtos é exibida na tela.

```java
public static void main(String[] args) {
    try {
        Socket socket = new Socket("localhost", 4321);
        ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
        ObjectInputStream in = new ObjectInputStream(socket.getInputStream());

        String login = "op1";
        String senha = "op1";
        out.writeObject(login);
        out.writeObject(senha);

        String response = (String) in.readObject();
        System.out.println(response);

        if ("Usuário logado com sucesso".equals(response)) {
            out.writeObject("L");

            List<Produto> produtos = (List<Produto>) in.readObject();
            for (Produto produto : produtos) {
                System.out.println("Produto: " + produto.getNome() + ", Preço: " + pr
            }
        } else {
            System.out.println("Falha na autenticação.");
        }

    } catch (IOException | ClassNotFoundException e) {
        System.err.println("Erro na comunicação com o servidor: " + e.getMessage());
    }
}
```

- Executando o CadastroServer e o CadastroCliente, para testar.



a) Como funcionam as classes Socket e ServerSocket?

Resposta:

- As classes Socket e ServerSocket em Java são usadas para comunicação de informações na rede. A classe ServerSocket escuta e aceita conexões de clientes, enquanto a classe Socket é usada pelos clientes para se conectar ao servidor. O servidor cria um ServerSocket e espera por conexões, e quando uma conexão é aceita, cria um Socket para comunicação.

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

Resposta:

- São importantes para a conexão com servidores porque permitem que diferentes serviços e aplicações troquem informações na rede. Cada porta identifica um serviço específico no servidor, permitindo que a informações permitindo que os dados cheguem ao destino correto.

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

Resposta:

- As classes ObjectInputStream e ObjectOutputStream em Java são usadas para ler e escrever objetos na entrada e saída, permitindo a transmissão de objetos entre programas, através de uma rede. Os objetos devem ser serializáveis por meio da classe Serializable para que possam ser convertidos em bytes e, depois reconstruídos no estado original.

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

Resposta:

- Foi possível garantir porque o cliente não acessa diretamente o banco de dados, o cliente interage com um servidor intermediário que utiliza JPA para realizar operações no banco de dados.

2º Procedimento | Servidor Completo e Cliente Assíncrono

Inserir neste campo, **de forma organizada**, todos os códigos do roteiro do 2º Procedimento da Atividade Prática, os resultados da execução do código e a Análise e Conclusão:

- Alteração no método run, do CadastroServer, para que agora ele receba comandos para funcionalidades de E - Entrada, S - Saída, L - Listagem, X - Encerrar programa.

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

        UsuarioOperador usuario = ctrlUsu.findUsuario(login, senha);

        if (usuario == null) {
            out.writeObject("Credenciais inválidas. Conexão encerrada.");
            s1.close();
            return;
        }

        out.writeObject("Usuário logado com sucesso");

        while (true) {

            String command = (String) in.readObject();

            if("X".equals(command)){
                out.close();
                in.close();
                s1.close();
            }

            if ("L".equals(command)) {
                List<Produto> produtos = ctrlProd.findProdutos();
                out.writeObject(produtos);
            } else if ("E".equals(command) || "S".equals(command)) {

                Integer idPessoa = (Integer) in.readObject();
```

```
                Integer idPessoa = (Integer) in.readObject();
                Pessoa pessoa = ctrlPessoa.findPessoa(idPessoa);

                Integer idProduto = (Integer) in.readObject();
                Produto produto = ctrlProd.findProduto(idProduto);

                Integer quantidade = (Integer) in.readObject();

                double valorUnitario = (in.readDouble());

                Movimento movimento = new Movimento(quantidade, command, valorUnitario, pe
                ctrlMov.create(movimento);

                if ("E".equals(command)) {
                    produto.setQuantidade(produto.getQuantidade() + quantidade);
                } else if ("S".equals(command)) {
                    produto.setQuantidade(produto.getQuantidade() - quantidade);
                }
                ctrlProd.edit(produto);

                out.writeObject("Movimento registrado com sucesso.");
            } else {

                out.writeObject("Comando desconhecido");
            }
        }

    } catch (IOException | ClassNotFoundException e) {
        System.err.println("Erro na comunicação com o cliente: " + e.getMessage());
        e.printStackTrace();
    }
```

- Criação do cliente assincrono, com Socket apontando para porta 4321, e o menu com as opções L – Listar, X – Finalizar, E – Entrada, S – Saída

```
try (
        Socket socket = new Socket("localhost", 4321); ObjectOutputStream out =

    out.writeObject("op1");
    out.writeObject("op1");

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

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

    Scanner scanner = new Scanner(System.in);

    while (true) {
        System.out.println("Menu:");
        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: ");
        String command = scanner.nextLine();

        out.writeObject(command);
        if ("X".equalsIgnoreCase(command)) {
            break;
        }
        if ("E".equalsIgnoreCase(command) || "S".equalsIgnoreCase(command)) {
            System.out.print("Id da pessoa: ");
            int idPessoa = Integer.parseInt(scanner.nextLine());
            out.writeObject(idPessoa);

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

```
            int idProduto = Integer.parseInt(scanner.nextLine());
            out.writeObject(idProduto);

            System.out.print("Quantidade: ");
            int quantidade = Integer.parseInt(scanner.nextLine());
            out.writeObject(quantidade);

            System.out.print("Valor unitário: ");
            double valorUnitario = Double.parseDouble(scanner.nextLine());
            out.writeDouble(valorUnitario);
            out.flush();
        }
    }
```

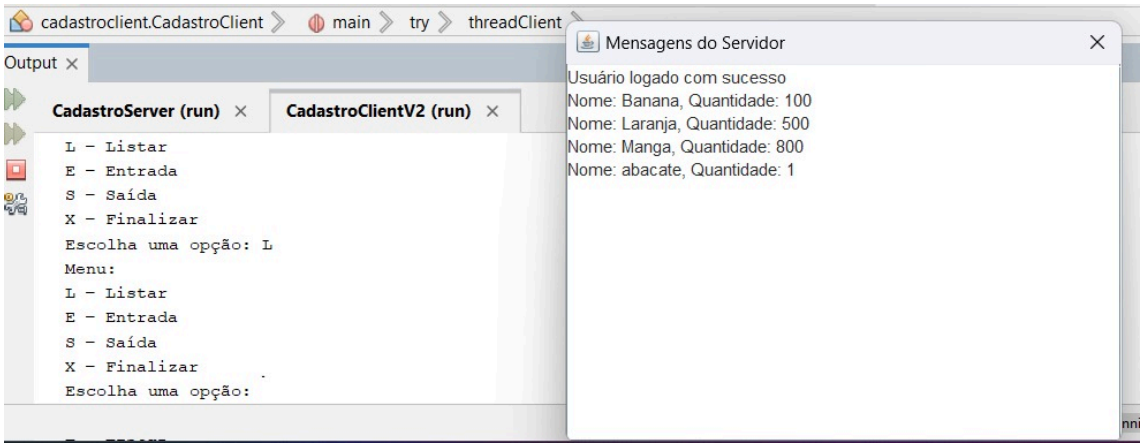
- Criação da classe SaidaFrame, que herda de JDialog para exibir as mensagens do servidor.

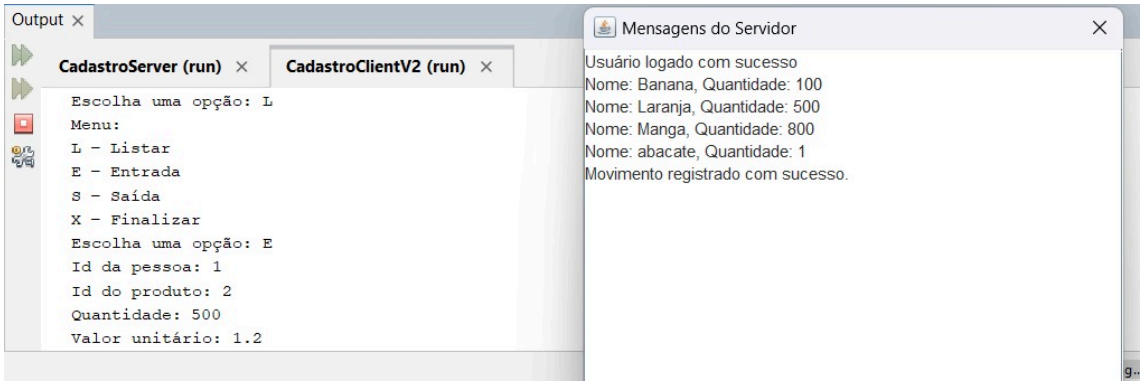
```
public class SaidaFrame extends JDialog {
    public JTextArea texto;

    public SaidaFrame(Frame owner) {
        super(owner, "Mensagens do Servidor", false);
        setBounds(100, 100, 400, 300);

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

        add(scrollPane);

        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
    }
```

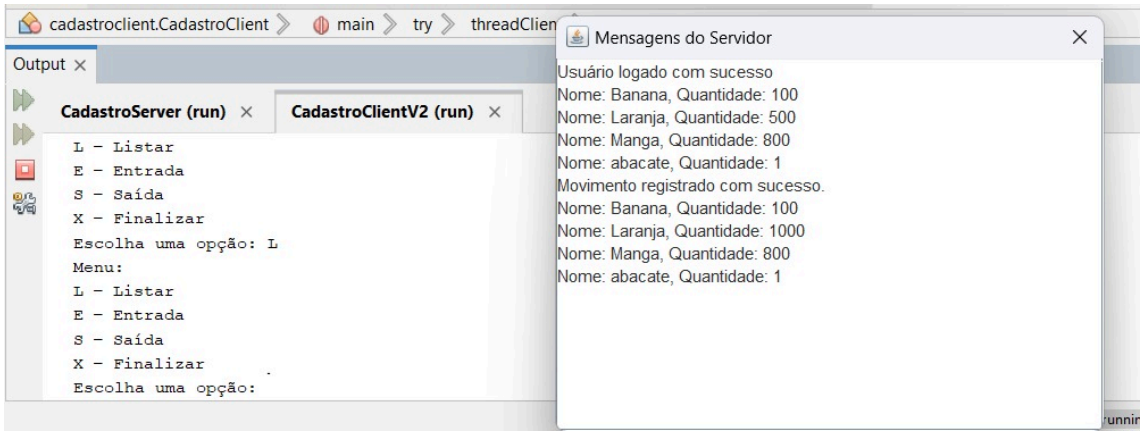
- Definição da ThreadCliente para realizar o preenchimento assíncrono das informações. No método run foi implementado um loop infinito de leitura no qual recebe os dados enviados pelo servidor para escrever na SaídaFrame.

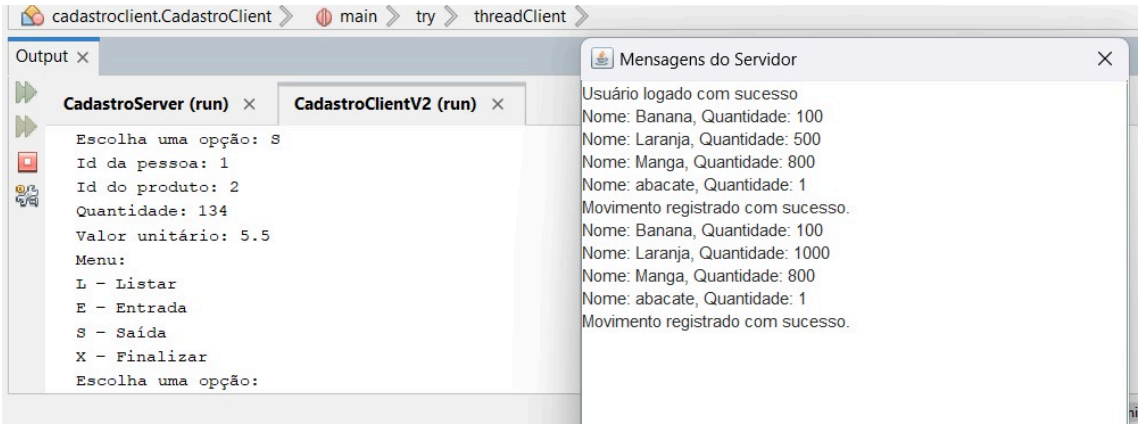
```
public ThreadClient(ObjectInputStream entrada, JTextArea textArea) {
    this.entrada = entrada;
    this.textArea = textArea;
}

@Override
public void run() {
    try {

        while (true) {

            Object obj = entrada.readObject();

            if (obj instanceof String) {
                String message = (String) obj;
                SwingUtilities.invokeLater(() -> textArea.append(message + "\n"));
            } else if (obj instanceof List) {
                List<?> list = (List<?>) obj;
                SwingUtilities.invokeLater(() -> {
                    for (Object item : list) {
                        if (item instanceof Produto) {
                            Produto produto = (Produto) item;
                            textArea.append("Nome: " + produto.getNome() + ", Quantidade: " + produto.getQuantidade() + "\n");
                        }
                    }
                });
            }

        }
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();
    }
}
```

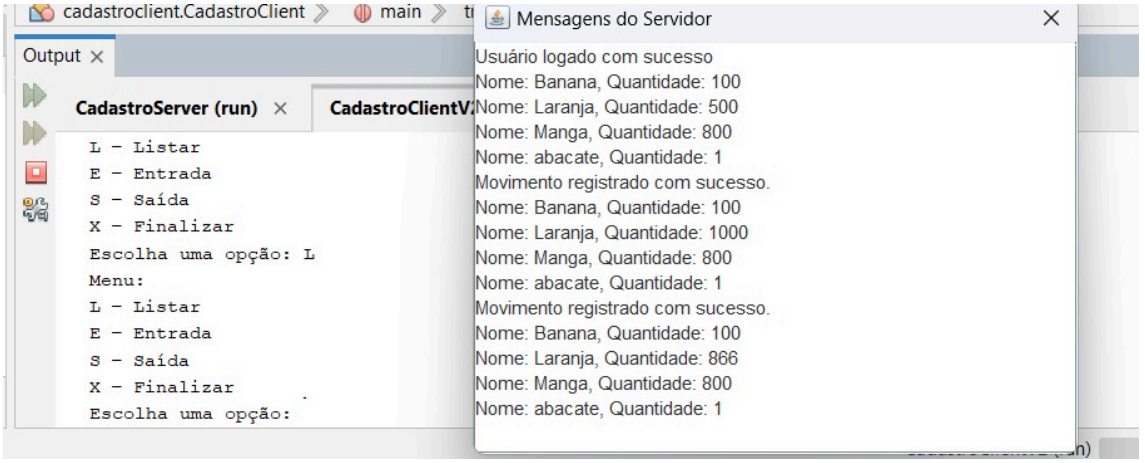
- Executando o CadastroServer e o CadastroCliente para testar as funcionalidades.











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

Resposta:

- As Threads podem ser usadas para tratamento assíncrono das respostas do servidor criando uma nova Thread para cada resposta recebida do cliente, ou seja, enquanto uma Thread processa a resposta, o programa principal continua executando outras tarefas.

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

Resposta:

- O método invokeLater serve para “agendar” a execução de um trecho de código do Swing, que é responsável pela interface gráfica. Usandoeste método, é possível garantir que atualizações na interface gráfica sejam feitas de forma segura e ordenada, evitando problemas.

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

Resposta:

- Os objetos são enviados e recebidos por meio de sockets usando o objeto ObjectOutputStream para escrever objetos na saída e ObjectInputStream para ler objetos na entrada.

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

Resposta:

- O comportamento síncrono bloqueia o cliente enquanto espera por respostas do servidor, o que pode tornar a aplicação menos responsiva. O comportamento assíncrono permite que o cliente continue executando outras tarefas enquanto aguarda as respostas.

Conclusão

No desenvolvimento do projeto CadastroServer e CadastroClient, implementamos um sistema de cadastro e controle de produtos utilizando Java com persistência JPA e comunicação via sockets. A estrutura do projeto incluiu entidades, controladores JPA, threads assíncronas para comunicação cliente-servidor, e interfaces gráficas para exibição de resultados. Enfrentamos desafios como gerenciamento correto de sockets e sincronização de threads para atualização das informações a serem mostradas para o cliente. O segundo procedimento abordou a implementação de um menu com operações de listagem, entrada e saída de produtos de forma segura e eficiente, destacando-se pela integração entre camadas de persistência e apresentação dos dados na interface gráfica.