



Vamos integrar sistemas

Israel dos Santos Hamdan D'Araujo , matricula: 202303592086

Jequié-Ba

Vamos integrar sistemas – 2023.1 – 2024.1

Objetivo da Prática

Realizar um sistema em Java modelo ANT, usando eclipselink como servidor, e socket de conexão para integrar sistema do Servidor e Sistema do cliente assíncrono

<https://github.com/IsraelHamdan/CadastroServer.git>

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:

CLASSE SERVIDOR:

```
package cadastroServer;  
  
import controller.PessoasJpaController;  
import controller.MovimentosJpaController;  
import controller.ProdutosJpaController;  
import controller.UsuariosJpaController;  
import java.io.IOException;  
import java.net.ServerSocket;  
import java.net.Socket;
```

```
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class CadastroServer {

    private static final Logger logger =
Logger.getLogger(CadastroServer.class.getName());

    private static EntityManagerFactory emf;
    private static ProdutosJpaController ctrl;
    private static UsuariosJpaController ctrlUsu;
    private static PessoasJpaController ctrlPessoas;
    private static MovimentosJpaController ctrlMov;
    private static CadastroThread ct;

    public static void main(String[] args) {

        emf =
Persistence.createEntityManagerFactory("CadastroServerPU");

        ctrl = new ProdutosJpaController(emf);
        ctrlUsu = new UsuariosJpaController(emf);
        ctrlMov = new MovimentosJpaController(emf);
        ctrlPessoas = new PessoasJpaController(emf);
```

```

        try (ServerSocket serverSocket = new
ServerSocket(4321)) {
            while (true) {
                try {
                    logger.log(Level.INFO, "Aguardando
conexão do cliente...");

                    Socket clientSocket =
serverSocket.accept();

                    if (clientSocket.isConnected()) {
                        ct = new CadastroThread(ctrl,
ctrlUsu, clientSocket);

                        ct.start();
                    } else {
                        logger.log(Level.SEVERE, "A conexão
com o cliente não foi estabelecida corretamente.");
                    }
                } catch (IOException ie) {
                    logger.log(Level.SEVERE, "Não foi
possível se conectar com o cliente", ie);
                }
            }
        } catch (IOException e) {

```

```

        logger.log(Level.SEVERE, "Não foi possível
iniciar o servidor", e);

    }

}
}

```

CLASSE CLIENTE:

```

package cadastroclient;

import java.io.*;

import java.net.Socket;

import java.util.logging.Level;
import java.util.logging.Logger;

public class CadastroClient {

    private static final Logger logger =
Logger.getLogger(CadastroClient.class.getName());

    public static void main(String[] args) {

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

            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

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

            PrintWriter out = new PrintWriter(socket.getOutputStream(), true)

        ) {

```

```

System.out.println("Insira o login:");

String login = console.readLine();

System.out.println("Insira a senha:");

String password = console.readLine();


out.println(login);

out.println(password);


String res = in.readLine();

System.out.println(res);


if ("Login bem-sucedido".equals(res)) {
    System.out.println("Digite 'L' para exibir os produtos");

    String command = console.readLine();

    out.println(command);

    String produtos;

    while ((produtos = in.readLine()) != null && !produtos.equals("END")) {
        System.out.println(produtos);
    }
} else {
    System.out.println("Falha no login. Tente novamente.");
}

} catch (IOException e) {

    logger.log(Level.SEVERE, "Não foi possível se conectar ao servidor",
e);

}

```

}

}

- a) Como funcionam as classes Socket e ServerSocket?

Resposta: O socket é uma extremidade de uma conexão bidirecional entre o cliente e o servidor rodando na rede, ele te permite se conectar a um endereço de IP portas específicos para envio e recebimento de dados. Já o ServerSocket é usado para criar um servidor que realizam as operações dos clientes ele espera por requisições em uma porta específica

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

Resposta: As portas são identificadores de processos que os servidores estão executando. Quando você se conecta com um servidor a porta deve ser especificada, isso é necessário para que o IP da sua máquina seja identifica sua maquina na rede e a porta o processo

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

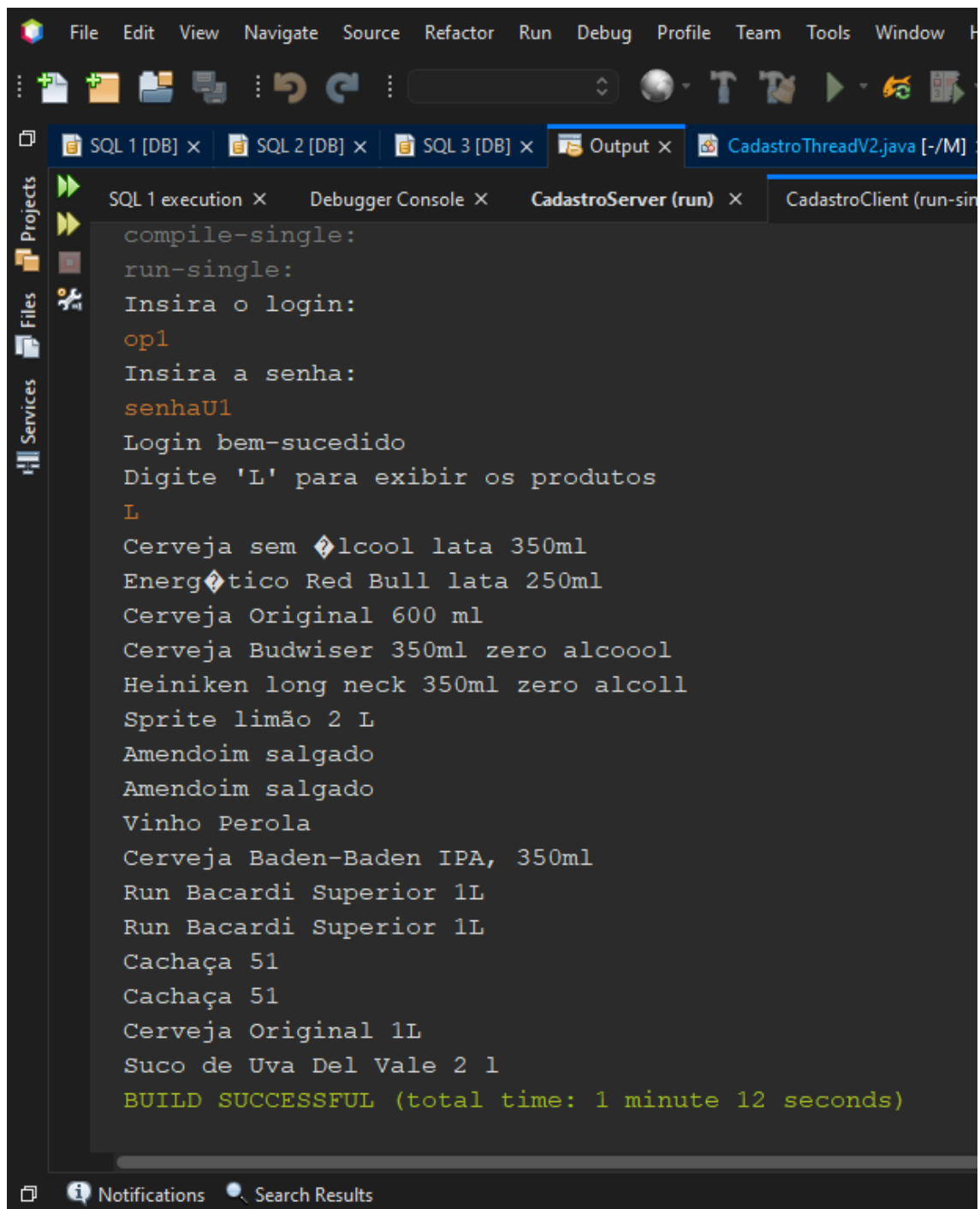
Resposta: ObjectInputStream serializa um objeto primitivo e gráficos de de objetos que foram serializados, ou seja, ele faz o fluxo de entrada. Já o ObjectOutputStream faz a serialização dos objetos sendo gravados como fluxo de entrada. Os objetos devem ser serializados para que possam ser convertidos em bytes, Isso faz com que eles possam ser transmitidos pela rede ou salvo em arquivos com o Serializable

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

Resposta: O cliente não tinha acesso ao banco de dados diretamente, ele somente recebia os dados vindos do Thread que se comunica com o servidor e o cliente.

Conclusão do primeiro procedimento: O primeiro procedimento foi bem simples de ser realizados, não foram encontradas dificuldades na realização do mesmo pois só foi necessário listar os produtos, observe que pela imagem que eu mandei, os produtos não são os mesmos do exemplo mostrado no Sway pois o meu banco de dados tem produtos diferentes

Resultado do primeiro procedimento



```
SQL 1 [DB] x SQL 2 [DB] x SQL 3 [DB] x Output x CadastroThreadV2.java [-/M]
SQL 1 execution x Debugger Console x CadastroServer (run) x CadastroClient (run-sin
compile-single:
run-single:
Insira o login:
op1
Insira a senha:
senhaU1
Login bem-sucedido
Digite 'L' para exibir os produtos
L
Cerveja sem álcool lata 350ml
Energético Red Bull lata 250ml
Cerveja Original 600 ml
Cerveja Budwiser 350ml zero alcool
Heiniken long neck 350ml zero alcool
Sprite limão 2 L
Amendoim salgado
Amendoim salgado
Vinho Perola
Cerveja Baden-Baden IPA, 350ml
Run Bacardi Superior 1L
Run Bacardi Superior 1L
Cachaça 51
Cachaça 51
Cerveja Original 1L
Suco de Uva Del Vale 2 l
BUILD SUCCESSFUL (total time: 1 minute 12 seconds)
```

2º Procedimento | Alimentando a Base

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:

Classe do servidor:

```
package cadastroServer;

import controller.PessoasJpaController;
import controller.MovimentosJpaController;
import controller.ProdutosJpaController;
import controller.UsuariosJpaController;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class CadastroServerV2 {

    private static final Logger logger =
Logger.getLogger(CadastroServerV2.class.getName());

    private static EntityManagerFactory emf;
    private static ProdutosJpaController ctrl;
    private static UsuariosJpaController ctrlUsu;
    private static PessoasJpaController ctrlPessoas;
    private static MovimentosJpaController ctrlMov;
    private static CadastroThreadV2 ct;
```



```

    public static void main(String[] args) {

        emf =
Persistence.createEntityManagerFactory("CadastroServerPU");

        ctrl = new ProdutosJpaController(emf);
        ctrlUsu = new UsuariosJpaController(emf);
        ctrlMov = new MovimentosJpaController(emf);
        ctrlPessoas = new PessoasJpaController(emf);

        try (ServerSocket serverSocket = new ServerSocket(4321))
        {
            while (true) {
                try {
                    logger.log(Level.INFO, "Aguardando conexão
do cliente...");

                    Socket clientSocket = serverSocket.accept();

                    if (clientSocket.isConnected()) {
                        ct = new CadastroThreadV2(ctrl, ctrlUsu,
ctrlMov, ctrlPessoas ,clientSocket);

                        ct.start();
                    } else {
                        logger.log(Level.SEVERE, "A conexão com
o cliente não foi estabelecida corretamente.");
                    }
                } catch (IOException ie) {

```

```

        logger.log(Level.SEVERE, "Não foi possível
se conectar com o cliente", ie);

    }

}

} catch (IOException e) {

    logger.log(Level.SEVERE, "Não foi possível iniciar o servidor", e);

}

}

}

```

Classe do cliente

```

package cadastroclient;

import java.io.*;

import java.net.Socket;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.swing.JOptionPane;

import javax.swing.SwingUtilities;


public class CadastroClientV2 {

    private SaidaFrame frame;

    private ThreadClient tc;


    private static final Logger LOGGER =
    Logger.getLogger(CadastroClientV2.class.getName());

```

```

public CadastroClientV2() {

    frame = new SaidaFrame();

}

private void display(String msg) {

    SwingUtilities.invokeLater(()                ->
frame.getTextArea().append(msg + "\n"));

}

private void connect() {

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

        PrintWriter out = new
PrintWriter(socket.getOutputStream(), true);

        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

        BufferedReader console = new BufferedReader(new
InputStreamReader(System.in))) {

        LOGGER.log(Level.INFO, "Cliente conectado ao
servidor");

        System.out.println("Insira o login:");

        String login = console.readLine();

        System.out.println("Insira a senha:");

        String password = console.readLine();

```

```

out.println(login);

out.println(password);


String res = in.readLine();

System.out.println(res);


if ("Login bem-sucedido".equals(res)) {

    display("Escolha uma opção");

    boolean running = true;

    tc = new ThreadClient(in, frame.getTextArea());

    tc.start();


    while (running) {

        System.out.println("Escolha uma opção:");

        System.out.println("L -> Listar | E ->
Entrada | S -> Saída | F -> Finalizar");

        String command = console.readLine();

        out.println(command);


        switch (command.toUpperCase()) {

            case "L":

                // Nada a fazer, ThreadClient irá
lidar com a resposta

                break;

```

```

        case "E":
        case "S":
            handleMoviment(out, console,
command.toUpperCase());

            break;
        case "F":
            running = false;
            out.println("F");
            break;
        default:
            System.out.println("Opção
inválida");

            break;
    }
}
} else {
    System.out.println("Falha no login: " + res);
}
} catch (IOException e) {
    LOGGER.log(Level.SEVERE, "Não foi possível se
conectar ao servidor", e);
}
}

```

```

    private void handleMoviment(PrintWriter out, BufferedReader
console, String command) throws IOException {

```

```
        String idProduto = JOptionPane.showInputDialog("Insira o  
id do produto");
```

```
        out.println(idProduto);
```

```
        String idPessoa = JOptionPane.showInputDialog("Insira o  
id da pessoa");
```

```
        out.println(idPessoa);
```

```
        String idUsuario = JOptionPane.showInputDialog("Insira o  
id do usuario");
```

```
        out.println(idUsuario);
```

```
        String quantidade = JOptionPane.showInputDialog("Insira  
a quantidade");
```

```
        out.println(quantidade);
```

```
        String preco = JOptionPane.showInputDialog("Insira o  
valor do produto");
```

```
        out.println(preco);
```

```
    }
```

```
public static void main(String[] args) {
```

```
    SwingUtilities.invokeLater(() -> {
```

```
        CadastroClientV2 client = new CadastroClientV2();
```

```
        client.connect();
```

```
    });
```

```
    }  
}
```

Classe CadastroThread

```
package cadastroServer;  
  
import model.Produtos;  
import model.Usuarios;  
import model.Movimentos;  
import controller.MovimentosJpaController;  
import controller.PessoasJpaController;  
import controller.ProdutosJpaController;  
import controller.UsuariosJpaController;  
import controller.exceptions.NonexistentEntityException;  
import controller.exceptions.PreexistingEntityException;  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
  
import java.io.PrintWriter;  
  
import java.net.Socket;  
import java.util.List;  
import java.util.logging.Level;  
import java.util.logging.Logger;
```

```

public class CadastroThreadV2 extends Thread {

    private final PessoasJpaController ctrlPessoa;

    private final ProdutosJpaController ctrl;

    private final UsuariosJpaController ctrlUsu;

    private final MovimentosJpaController ctrlMov;


    private final Socket s1;

    private static final Logger logger =
Logger.getLogger(CadastroThreadV2.class.getName());


    public CadastroThreadV2(ProdutosJpaController ctrl,
UsuariosJpaController ctrlUsu,

        MovimentosJpaController ctrlMov,
PessoasJpaController ctrlPessoa, Socket s1) {

        this.ctrl = ctrl;

        this.ctrlUsu = ctrlUsu;

        this.ctrlMov = ctrlMov;

        this.ctrlPessoa = ctrlPessoa;


        this.s1 = s1;
    }


    private void sellingProduct(BufferedReader in, PrintWriter
out, String command) throws IOException {

        Integer idProduto = Integer.parseInt(in.readLine());

        Produtos produto = ctrl.findProduto(idProduto);

```



```

Movimentos movimentos = new Movimentos();

movimentos.setIdProduto(produto);


Integer idPessoa = Integer.parseInt(in.readLine());
movimentos.setIdPessoa(ctrlPessoa.findPessoa(idPessoa));


Integer idUser = Integer.parseInt(in.readLine());
movimentos.setIdUsuario(ctrlUsu.findUsuario(idUser));


movimentos.setTipo(command);


int quantidade = Integer.parseInt(in.readLine());
movimentos.setQuantidade(quantidade);


Float preco = Float.parseFloat(in.readLine());
movimentos.setValorUnitario(preco);


produto.setQuantidade(produto.getQuantidade()
quantidade);
try {
    ctrl.edit(produto);
    ctrlMov.create(movimentos);
    out.println("Produto vendido com sucesso");
    outputMovimment(out);
}

```

```

        } catch (NonexistentEntityException |
PreexistingEntityException ex) {

            logger.log(Level.SEVERE, "Erro ao vender o produto",
ex);

            out.println("Erro ao vender o produto");
        }
    }
}

```

```

    private void buyingProduct(BufferedReader in, PrintWriter
out, String command) throws IOException,
NonexistentEntityException {

        Integer idProduto = Integer.parseInt(in.readLine());

        Produtos produto = ctrl.findProduto(idProduto);

        Movimentos movimentos = new Movimentos();

        movimentos.setIdProduto(produto);

        Integer idPessoa = Integer.parseInt(in.readLine());

        movimentos.setIdPessoa(ctrlPessoa.findPessoa(idPessoa));

        Integer idUser = Integer.parseInt(in.readLine());

        movimentos.setIdUsuario(ctrlUsu.findUsuario(idUser));

        movimentos.setTipo(command);

        int quantidade = Integer.parseInt(in.readLine());

        movimentos.setQuantidade(quantidade);
    }
}

```

```

        Float preco = Float.parseFloat(in.readLine());

        movimentos.setValorUnitario(preco);

        produto.setQuantidade(produto.getQuantidade()
                                +
                                quantidade);

        try {

            ctrl.edit(produto);

            ctrlMov.create(movimentos);

            out.println("Produto comprado com sucesso");

            outputMovimment(out);

        } catch (NonexistentEntityException |
PreexistingEntityException ex) {

            logger.log(Level.SEVERE, "Erro ao comprar o
produto", ex);

            out.println("Erro ao comprar o produto");

        }

    }

    private void ListingProducts(PrintWriter out) {

        List<Produtos> produtos = ctrl.findProdutoEntities();

        for (Produtos produto : produtos) {

            out.println("Produtos: " + produto.getNome() + " -
Quantidade: " + produto.getQuantidade());

        }
    }

```

```

        out.println("END");
    }

    private void outputMovimment(PrintWriter out) {

        List<Movimentos>                movimentos                =
ctrlMov.findMovimentoEntities();

        int indice = movimentos.size() -1;

        String tipo = movimentos.get(indice).getTipo();

        String                loginU                =
movimentos.get(indice).getIdUsuario().getLogin() ;

        String                produto                =
movimentos.get(indice).getIdProduto().getNome();

        String                pessoa                =
movimentos.get(indice).getIdPessoa().getNome();

        Float valor = movimentos.get(indice).getQuantidade() *
movimentos.get(indice).getIdProduto().getPreco();

        String msg = String.format("Movimento: "+"Tipo: %s |
Usuario: %s | Produto: %s | Pessoa: %s | valor %.2f" , tipo,

                loginU, produto,pessoa,valor

    ) ;

        out.println("begin");

        out.println(msg);

        out.flush();

        out.println("END");

        System.out.println(msg);
    }

```

```

    }

    @Override
    public void run() {
        try (BufferedReader in = new BufferedReader(new
InputStreamReader(s1.getInputStream()));
            PrintWriter out = new
PrintWriter(s1.getOutputStream(), true)) {

            logger.log(Level.INFO, "Thread iniciada para
comunicação com o cliente");

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

            Usuarios usuario =
ctrlUsu.findUsuarioByLogin(login);

            if (usuario == null ||
!usuario.getSenha().equals(senha)) {
                out.println("Login inválido");
                logger.log(Level.SEVERE, "Não foi possível fazer
login");

                return;
            }

            out.println("Login bem-sucedido");

```

```
boolean running = true;

while (running) {

    String command = in.readLine();

    if (command == null) break;

    logger.log(Level.INFO, "Comando recebido: " +
command);

    switch (command.toUpperCase()) {

        case "L":

            ListingProducts(out);

            break;

        case "E":

            buyingProduct(in, out, command);

            break;

        case "S":

            sellingProduct(in, out, command);

            break;

        case "F":

            running = false;

            break;

        default:

            logger.log(Level.WARNING, "Comando
desconhecido: " + command);
```

```

        out.println("Comando desconhecido: " +
command);

        break;
    }
}

} catch (IOException e) {

    logger.log(Level.SEVERE, "Erro ao se comunicar com o
cliente: ", e);

} catch (NonexistentEntityException ex) {

    logger.log(Level.SEVERE, "Erro ao editar entidade",
ex);

} finally {

    try {

        if (s1 != null && !s1.isClosed()) {

            s1.close();

        }

    } catch (Exception e) {

        logger.log(Level.SEVERE, "Erro ao fechar o
socket", e);

    }

}

}
}

```

Classe ThreadClient

```
package cadastroclient;
```

```

import java.io.BufferedReader;

import java.io.IOException;

import javax.swing.JTextArea;

import javax.swing.SwingUtilities;


public class ThreadClient extends Thread {

    private final BufferedReader in;

    private final JTextArea textArea;


    public ThreadClient(BufferedReader in, JTextArea textArea) {

        this.in = in;

        this.textArea = textArea;

    }


    @Override

    public void run() {

        try {

            String mensagem;

            while ((mensagem = in.readLine()) != null) {

                processarMensagem(mensagem);

            }

        } catch (IOException e) {

            System.out.println("Erro ao ler mensagem do
servidor: " + e.getMessage());

```



```

    } finally {
        if (in != null) {
            try {
                in.close();
            } catch (IOException e) {
                System.out.println("Erro ao fechar
BufferedReader: " + e.getMessage());
            }
        }
    }
}

```

```

private void processarMensagem(String mensagem) {
    SwingUtilities.invokeLater(() -> {
        if (mensagem.startsWith("Produtos: ")) {
            exibirListaDeProdutos(mensagem.substring("Produtos:
".length()));
        } else if (mensagem.startsWith("Movimento: ")) {
            exibirMovimento(mensagem.substring("Movimento:
".length()));
        } else {
            exibirMensagem(mensagem);
        }
    });
}

```

```

private JTextArea exibirListaDeProdutos(String produtosInfo)
{
    textArea.append("Lista de Produtos:\n");
    String[] produtos = produtosInfo.split(";");
    for (String produto : produtos) {
        textArea.append(produto + "\n");
    }
    textArea.append("-----\n");
    return textArea;
}

```

```

private JTextArea exibirMovimento(String movimentoInfo) {
    textArea.append("Detalhes do Movimento:\n");
    textArea.append(movimentoInfo + "\n");
    textArea.append("-----\n");
    return textArea;
}

```

```

private JTextArea exibirMensagem(String mensagem) {
    textArea.append(mensagem + "\n");
    return textArea;
}
}

```

Classe SaidFrame

```
package cadastrclient;

import java.awt.BorderLayout;
import javax.swing.*;

public class SaidaFrame extends JDialog {

    private JTextArea texto;

    public SaidaFrame() {

        setTitle("Movimento");

        setBounds(100, 100, 450, 300);

        setModal(false);

        texto = new JTextArea();
        texto.setEditable(false);
        add(new JScrollPane(texto), BorderLayout.CENTER);
        setVisible(true);
    }

    public JTextArea getTextArea() {

        return texto;
    }

}
```

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

Resposta: As threads permitem que o programa espere por uma resposta enquanto ele renderiza uma interface gráfica, isso é fundamental quando precisamos renderizar uma interface gráfica pois ela requer responsividade e precisa estar executando continuamente pois processam várias conexões simultaneamente.

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

Resposta: É usada para gerenciar todos os eventos e atualizações da interface gráfica para que os eventos de uma interface gráfica não se atropelam (concorrência), ela garante que tudo rode no seu devido tempo pois usa um `Event Dispatch Thread`

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

Resposta: Os objetos são transmitidos através de sockets

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

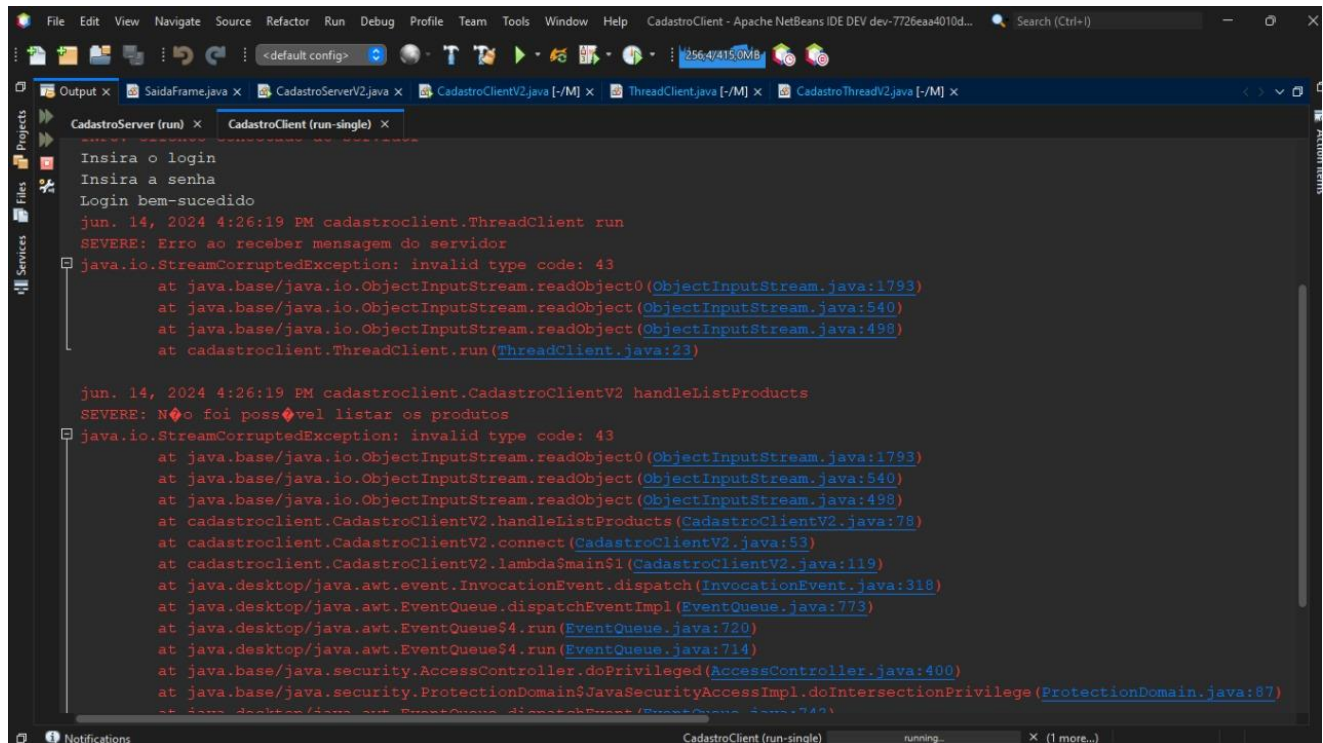
Resposta:

- Comportamento síncrono: o cliente espera que cada operação termine antes de continuar a próxima, isso pode bloquear o processamento de outras partes do programa, pois a thread não continua enquanto não recebe a resposta.
- Comportamento assíncrono: O cliente não precisa esperar uma resposta do servidor, ele continua processando as informações, a thread não é parada e quando a resposta é exibida ele toma a ação necessária

Observe que os tópicos acima seguem exatamente o que está na Atividade Prática exigida.

Conclusão do segundo procedimento: O procedimento já foi bem mais difícil de ser realizado por causa dos Objetos de entrada e saída, eles travaram de mais

o desenvolvimento do código, pois gerava erros a todo momento, eu não consegui utilizar eles no segundo procedimento, como pode ver pela imagem, acredito que os problemas eram causados pelo cast de variáveis que sempre tinha que fazer pra converter algo em string ou alguma outra tipagem

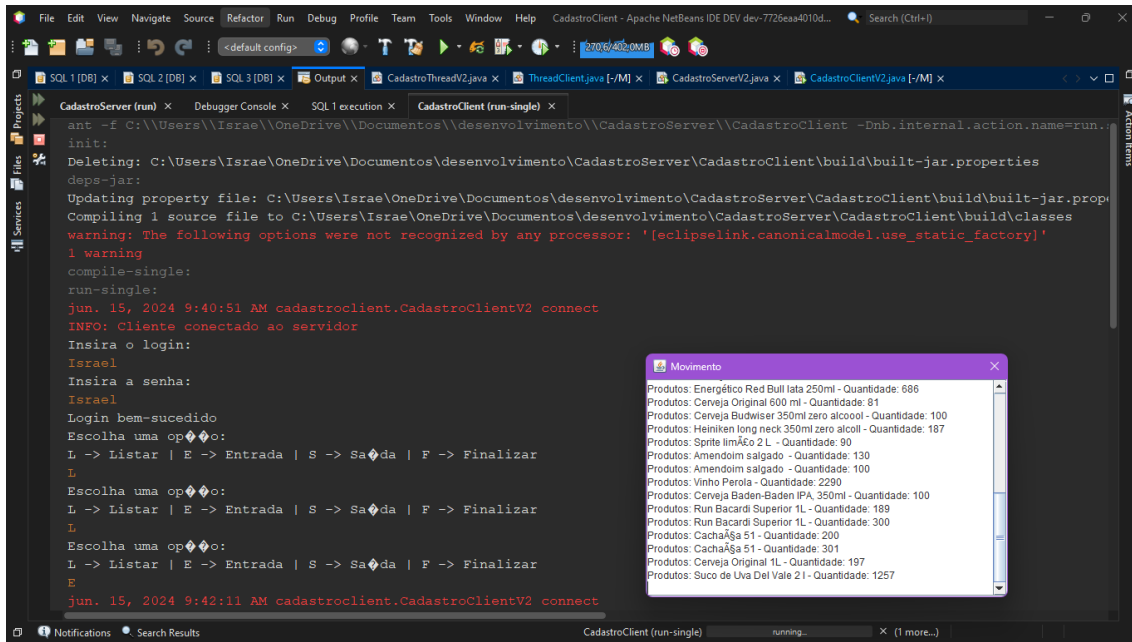


```
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help CadastroClient - Apache NetBeans IDE DEV-7726aa4010d... Search (Ctrl-I)
Output x SaidFrame.java x CadastroServerV2.java x CadastroClientV2.java [-/M] x ThreadClient.java [-/M] x CadastroThreadV2.java [-/M] x
CadastroServer (run) x CadastroClient (run-single) x
Insira o login
Insira a senha
Login bem-sucedido
jun. 14, 2024 4:26:19 PM cadastroclient.ThreadClient run
SEVERE: Erro ao receber mensagem do servidor
java.io.StreamCorruptedException: invalid type code: 43
    at java.base/java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1793)
    at java.base/java.io.ObjectInputStream.readObject(ObjectInputStream.java:540)
    at java.base/java.io.ObjectInputStream.readObject(ObjectInputStream.java:498)
    at cadastroclient.ThreadClient.run(ThreadClient.java:23)

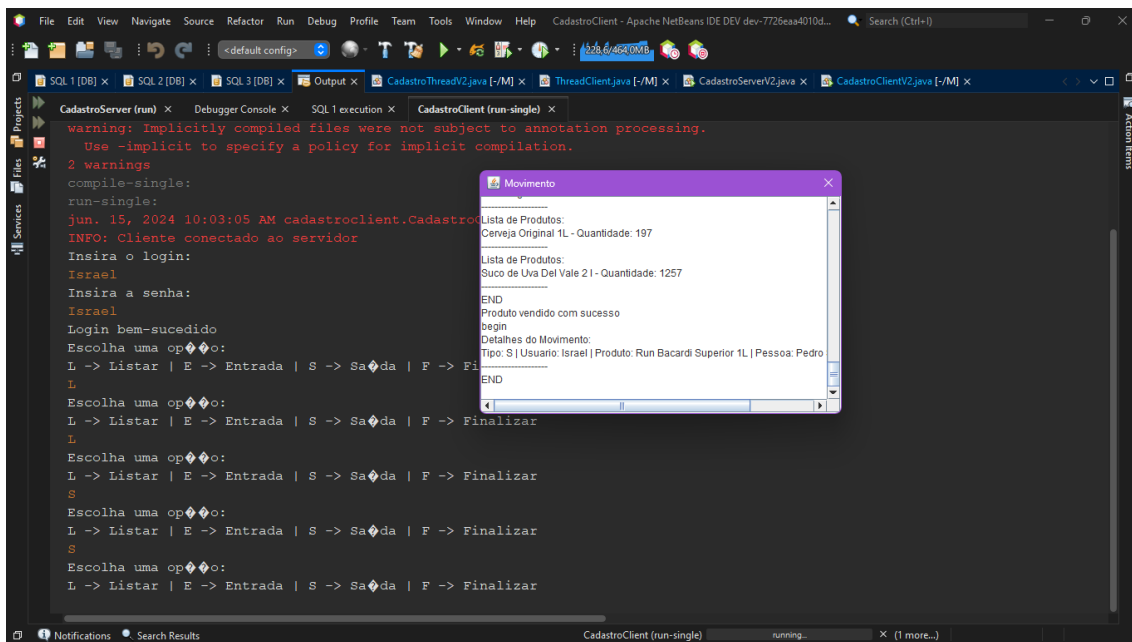
jun. 14, 2024 4:26:19 PM cadastroclient.CadastroClientV2 handleListProducts
SEVERE: Não foi possível listar os produtos
java.io.StreamCorruptedException: invalid type code: 43
    at java.base/java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1793)
    at java.base/java.io.ObjectInputStream.readObject(ObjectInputStream.java:540)
    at java.base/java.io.ObjectInputStream.readObject(ObjectInputStream.java:498)
    at cadastroclient.CadastroClientV2.handleListProducts(CadastroClientV2.java:78)
    at cadastroclient.CadastroClientV2.connect(CadastroClientV2.java:53)
    at cadastroclient.CadastroClientV2.lambda$main$1(CadastroClientV2.java:119)
    at java.desktop/java.awt.event.InvocationEvent.dispatch(InvocationEvent.java:318)
    at java.desktop/java.awt.EventQueue.dispatchEventImpl(EventQueue.java:773)
    at java.desktop/java.awt.EventQueue$4.run(EventQueue.java:720)
    at java.desktop/java.awt.EventQueue$4.run(EventQueue.java:714)
    at java.base/java.security.AccessController.doPrivileged(AccessController.java:400)
    at java.base/java.security.ProtectionDomain$JavaSecurityAccessImpl.doIntersectionPrivilege(ProtectionDomain.java:87)
    at java.desktop/java.awt.EventQueue.dispatchEvent(EventQueue.java:742)
```

A única solução encontrada foi usar o `BufferedReader` para entrada e o `Printwriter` na saída, se não fosse isso, eu não ia conseguir entregar a missão a tempo

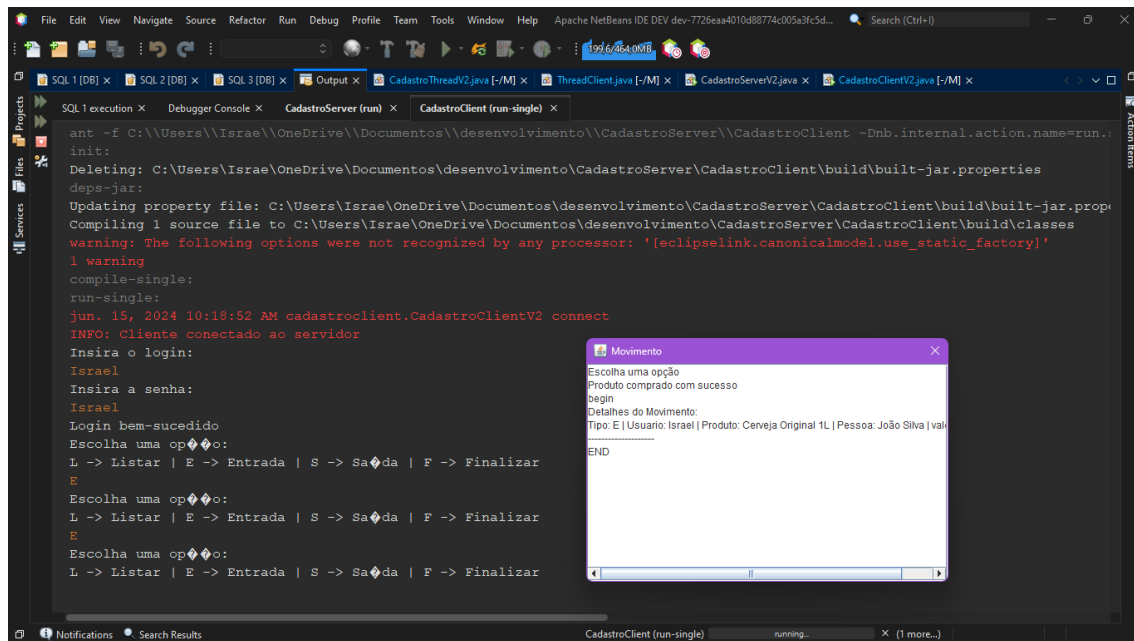
Resultado da Listagem de produtos



Resultado da saída



Resultado da entrada de produtos



Conclusão

Como já referido anteriormente, a primeira parte da missão foi extremamente simples de fazer, tendo em vista o pesadelo que foi a missão 4, mas como o futuro é uma caixinha de surpresas, a segunda parte da missão foi muito difícil de realizar por conta dos erros relacionados ao `ObjectOutputStream` e `ObjectInputStream` e os inúmeros cast de variáveis. O projeto não está rodando 100 fluido, ainda tem alguns bugs, mas foi o que eu consegui fazer, com ajuda de alguns colegas eu consegui resolver alguns bugs, e consultando na internet foi o jeito que eu consegui resolver essa missão.