**Universidade Salvador (UNIFACS)**

Nome 1: Gabriel Antônio Lopes de Castro RA: 1272023100

Nome 2: João Amaral Lantyer RA: 1271919682

Nome 3: Nelson Anísio Nascimento do Carmo RA: 1272018392

Nome 4: Paulo Sérgio Moraes de Oliveira Filho RA: 1272022847

Nome 5: Rafael Edmundo da Silva Neto RA: 12720111236

Instituição: Universidade Salvador (UNIFACS)

Orientador: Adailton de Jesus Cerqueira Junior

Disciplina: Sistemas distribuídos e mobile

Atividade: Relatório da A3 de Sistemas distribuídos e mobile



Salvador

2023

**SUMÁRIO**

1. **INTRODUÇÃO3**
2. **DESENVOLVIMENTO4**
3. **ALGORITMO DE ELEIÇÃO UTILIZADO31**
4. **CONCLUSÃO E INSTRUÇÕES32**

**1. INTRODUÇÃO**

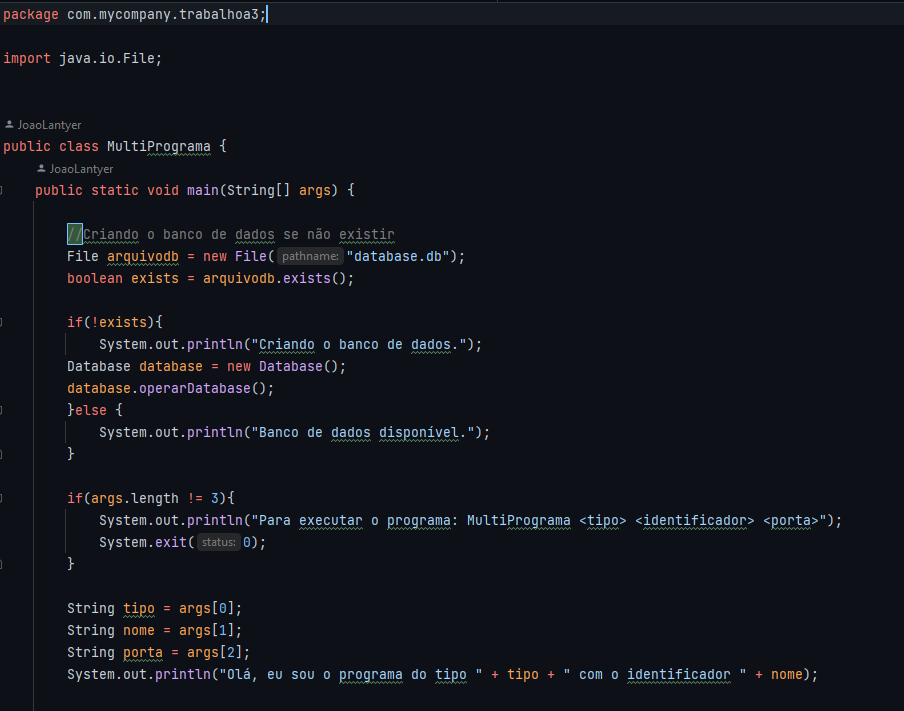
Este relatório apresenta o Projeto A3 de Sistemas Distribuídos e mobile, no qual desenvolvemos um aplicativo em Java utilizando sockets para simular a coleta de dados de vendas de uma rede de lojas. O aplicativo permite que os vendedores insiram os dados da venda, como seu nome, o nome do produto, a data da venda e a quantidade de produtos vendidos (o valor total é calculado com base no preço do produto). Estes serão armazenados no banco de dados (Utilizamos o SQLite neste projeto).

Além disso, o aplicativo oferece funcionalidades para os gerentes, que podem consultar diversas informações, como o total de vendas de um vendedor específico, o total de vendas de um determinado produto, o total de vendas de um produto dentro de um período de tempo específico, além de identificar o melhor vendedor e o produto mais vendido.

O servidor do aplicativo aguarda as solicitações dos clientes, identificando-as por meio de códigos específicos, e retorna mensagens de sucesso caso a solicitação seja bem-sucedida, ou mensagens de erro em caso contrário.

**2. DESENVOLVIMENTO**

A classe “MultiPrograma”:



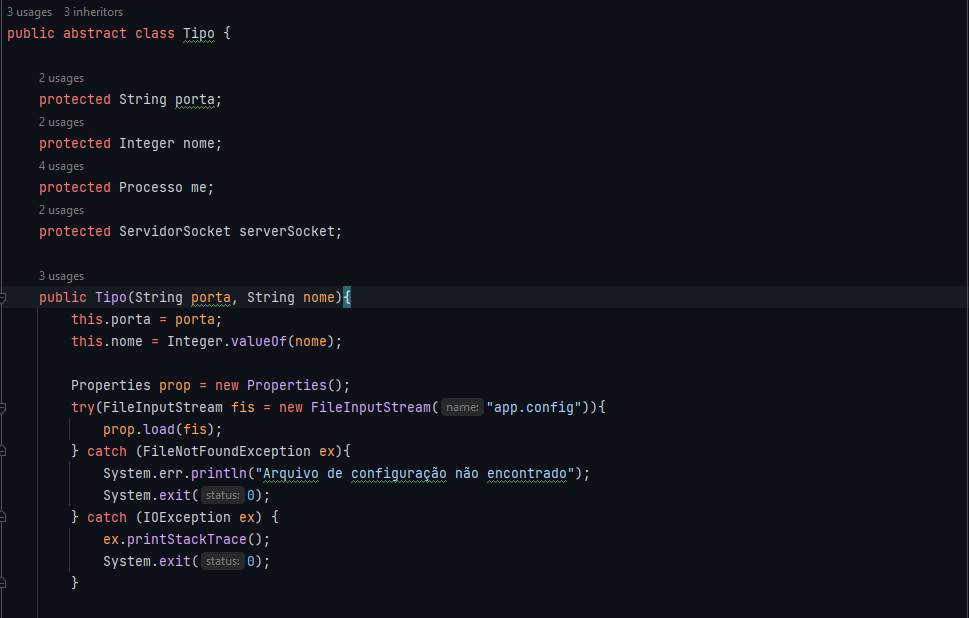


Começaremos pela classe “MultiPrograma” que serve como uma porta de entrada para o nosso aplicativo. Ela verifica se o arquivo do banco de dados chamado "database.db" existe. Se o arquivo não existir, ele executa o método da classe Database que cria o banco de dados e insere os dados necessários. Depois ela verifica se o número de argumentos passados no arquivo “start.bat” da linha de comando não é igual a 3. Se não houver exatamente três argumentos, ele imprime uma mensagem indicando o uso correto do programa e é fechado.

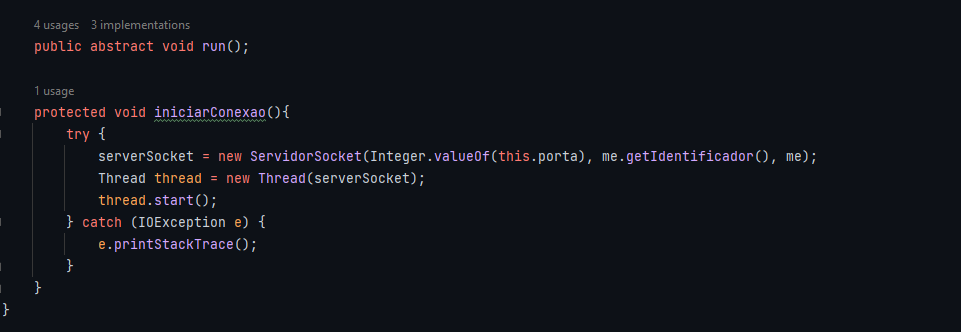
Se forem fornecidos três argumentos, eles são atribuídos às variáveis tipo, nome e porta, respectivamente.

O código então utiliza uma instrução switch baseada no valor da variável “tipo” para instanciar e executar diferentes tipos de programas. Dependendo do valor de tipo, uma instância das classes Gerente, Vendedor ou Servidor é criada e o método run é chamado.

A classe “Tipo”:



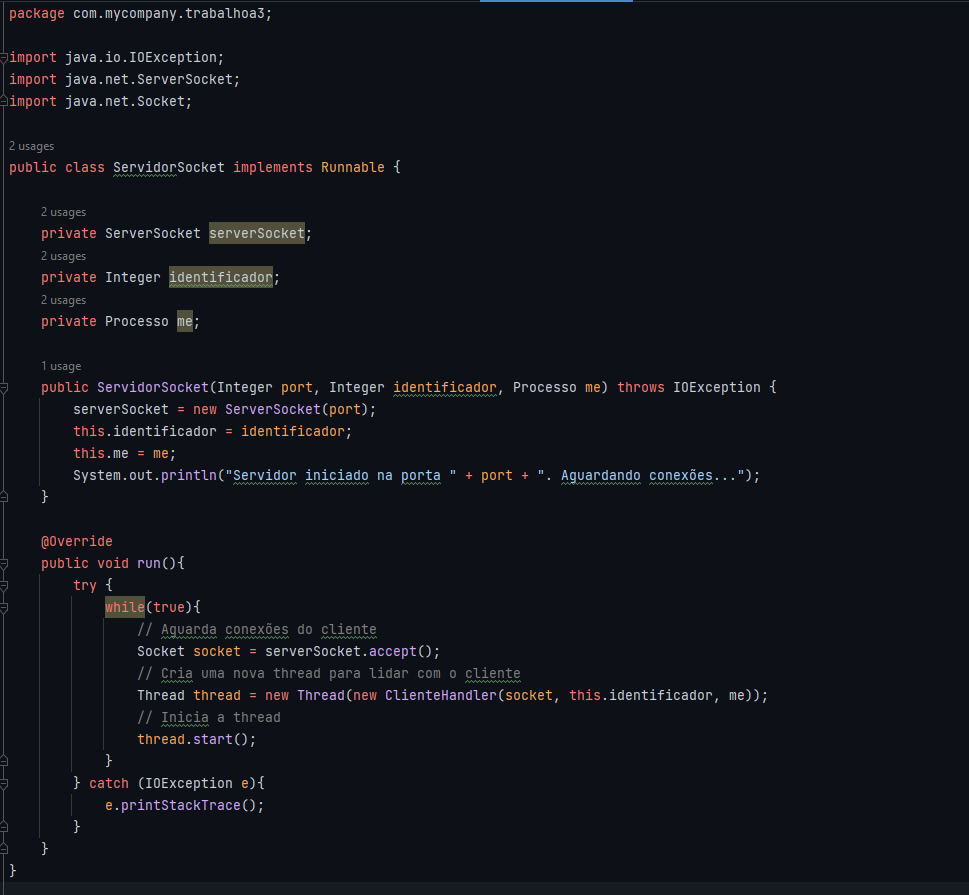




A classe “Tipo” é uma classe abstrata que é herdada pelo “Gerente” e pelo “Vendedor”, ela possui dois atributos (porta e nome). Além disso ela faz a checagem da existência do arquivo “app.config” na pasta raíz do projeto. O arquivo “app.config” denomina quais portas e hosts poderão ser usados pelo programa, além de limitar quantos processos poderão ser iniciados.

A classe “Tipo” possui dois métodos, o método “run()” (que é executado assim que os processos “Gerente” e “Vendedor” são iniciados) e o método “iniciarConexao()” (que inicia uma conexão de um processo com o servidor).

A classe “ServidorSocket”:



A classe "ServidorSocket" implementa a interface "Runnable" que possibilita a criação de Threads. Essa classe é responsável por criar um servidor de sockets que aceita conexões de clientes e delega o processamento dessas conexões para threads separadas.

O construtor da classe "ServidorSocket" recebe três parâmetros: "port" (porta na qual o servidor será iniciado), "identificador" (o identificador numérico do processo) e "me" (uma instância da classe "Processo" que virá a ser conectado).

No construtor, é criada uma instância de "ServerSocket" que será associada à porta especificada. Essa instância é atribuída à variável "serverSocket".

Dentro do construtor, existe uma função que exibe uma mensagem indicando que o servidor foi iniciado na porta especificada e está aguardando conexões.

Dentro do método "run()", há um loop infinito "while(true)" que aguarda continuamente conexões de clientes.

Quando uma conexão é estabelecida, o método "accept()" do "serverSocket" é chamado, bloqueando a execução até que um cliente se conecte. Quando isso acontece, um novo objeto "Socket" é retornado, representando a conexão com o cliente.

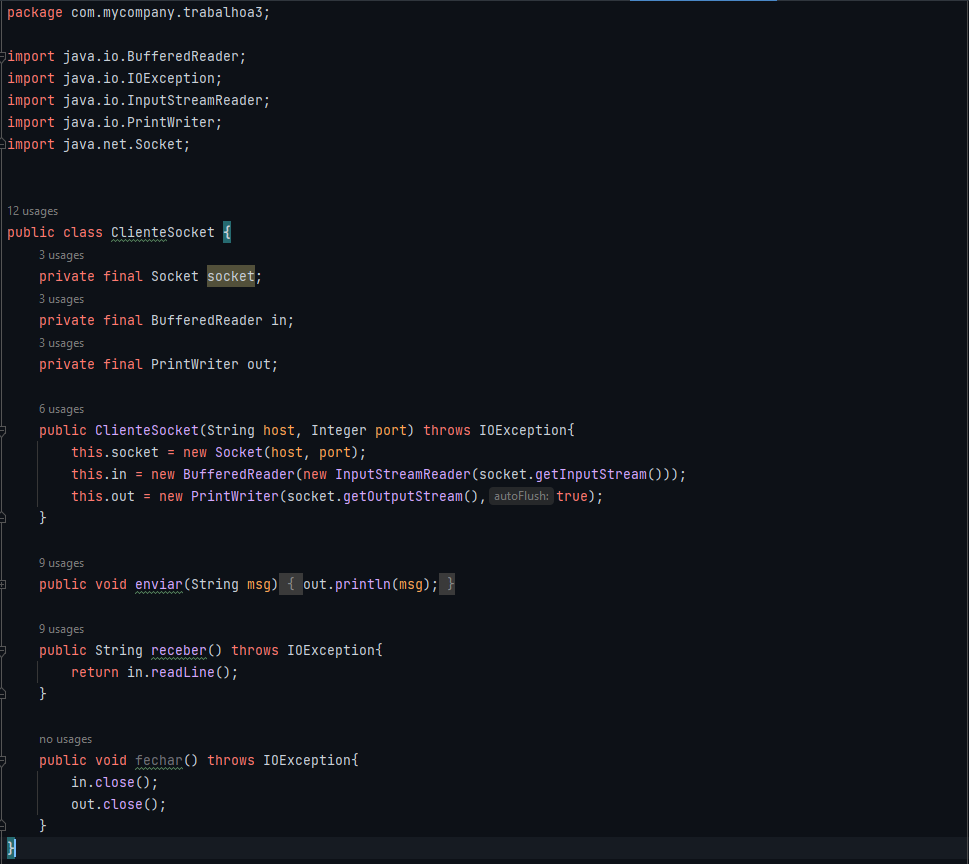
Em seguida, é criada uma nova thread e passado o objeto "Socket", o identificador e a instância do "Processo" para o construtor da classe "ClienteHandler". Essa classe é responsável por manipular as operações relacionadas ao cliente conectado.

A nova thread é iniciada chamando o método "start()", o que faz com que a execução do código no método "run()" da classe "ClienteHandler" seja realizada em paralelo com a execução do código no método "run()" da classe "ServidorSocket".

O loop no método "run()" da classe "ServidorSocket" continua a aguardar e aceitar novas conexões de clientes, criando uma nova thread para cada cliente conectado.

Se ocorrer uma exceção do tipo "IOException" durante o processo de aceitação de conexões ou criação das threads de manipulação de clientes, a exceção será impressa, mas o servidor continuará em execução.

A Classe “ClienteSocket”:



O código fornecido representa uma classe chamada ClienteSocket que é responsável por criar e gerenciar uma conexão de socket com o servidor.

O construtor da classe ClienteSocket recebe dois parâmetros: host (host do servidor) e port (a porta do servidor).

No construtor, é criado um objeto Socket com o endereço e porta especificados. Esse objeto representa a conexão de socket com o servidor.

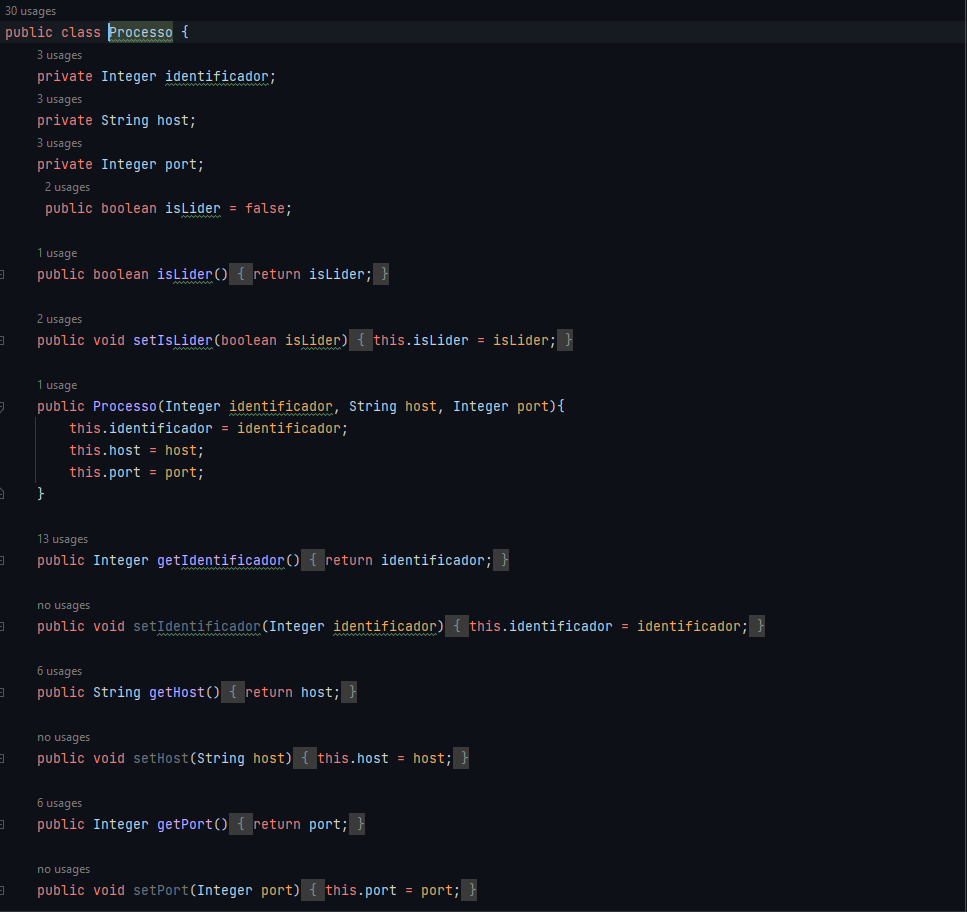
Também no construtor, são criados objetos BufferedReader e PrintWriter para a leitura e escrita de dados na conexão do socket. O BufferedReader é usado para ler dados do servidor e o PrintWriter é usado para enviar dados para o servidor.

O método enviar(String msg) é responsável por enviar uma mensagem para o servidor. Ele escreve a mensagem no PrintWriter associado ao socket, que será enviada para o servidor.

O método receber() é responsável por receber uma mensagem do servidor. Ele lê uma linha de texto do BufferedReader associado ao socket, que contém a mensagem recebida do servidor.

O método fechar() é responsável por fechar os recursos associados à conexão de socket, incluindo o BufferedReader, o PrintWriter e o próprio socket.

A Classe “Processo”:



A classe “Processo” serve como modelo de dados para representar um processo no nosso aplicativo.

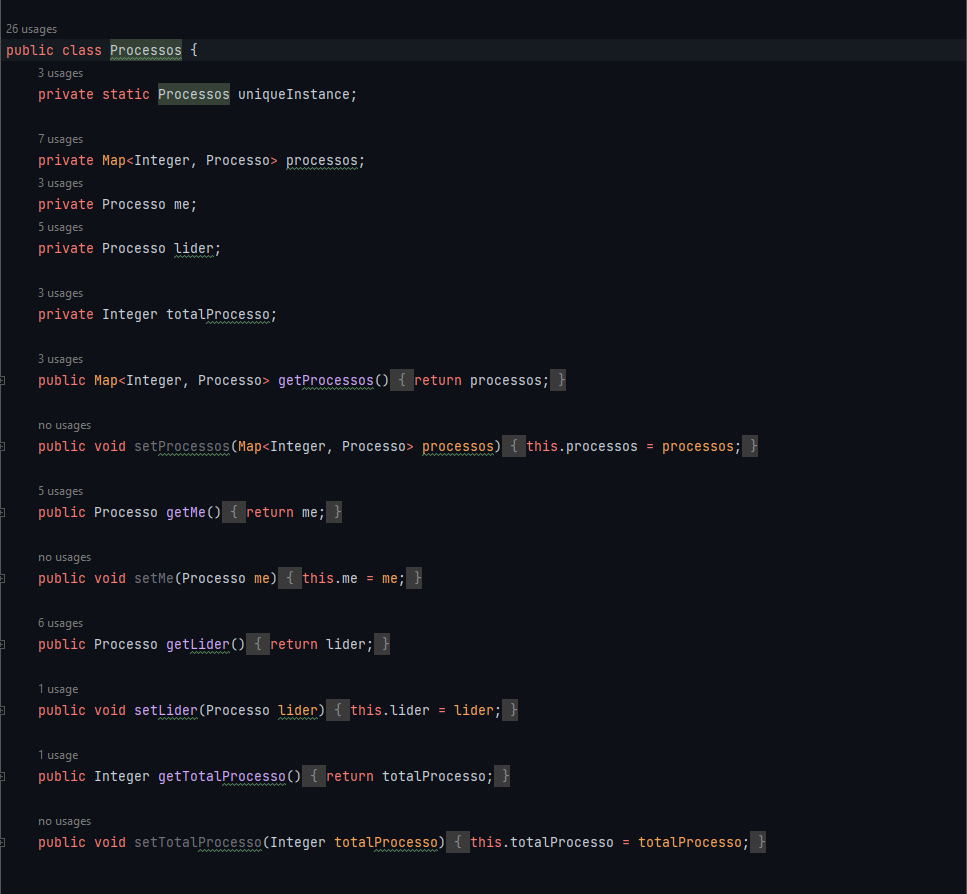
A classe Processo possui três atributos: identificador, que é um número inteiro que identifica o processo; host, que é uma string que representa o nome do host onde o processo está sendo executado; e port, que é um número inteiro que representa a porta associada ao processo.

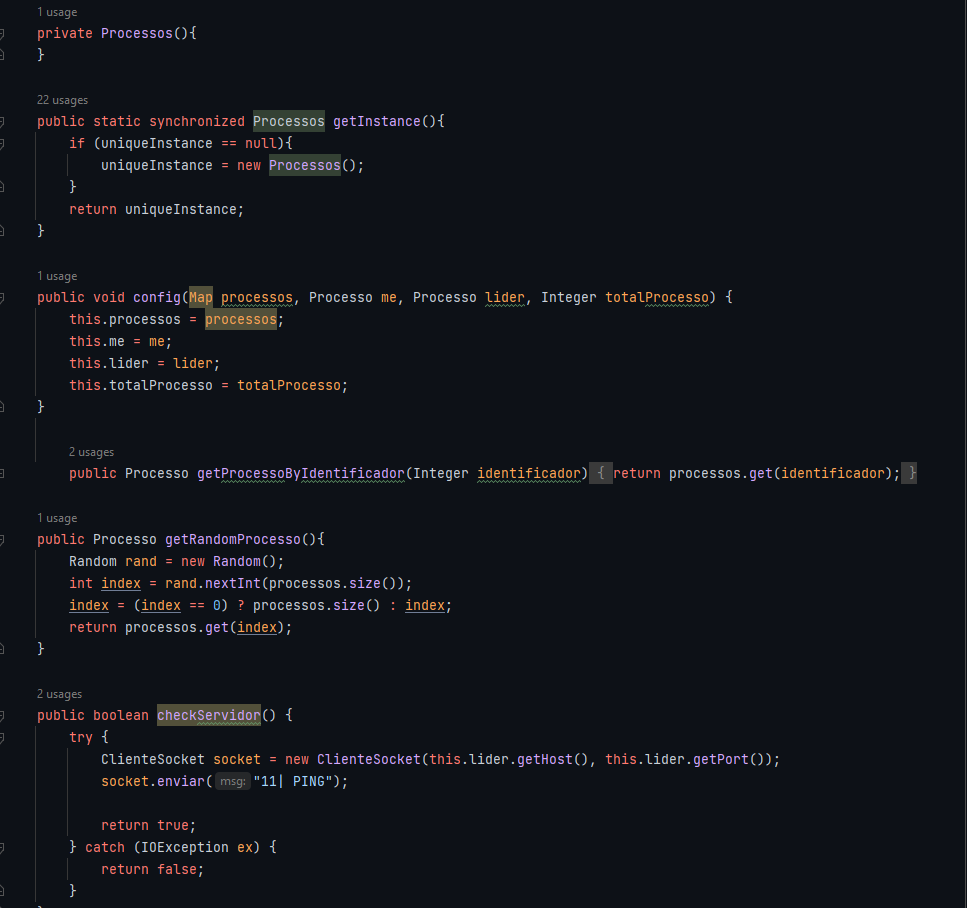
A classe possui um atributo adicional chamado isLider, que é um booleano utilizado para indicar se o processo é um líder. O atributo isLider possui os métodos isLider() e setIsLider(), que permitem acessar e modificar o valor desse atributo.

O construtor da classe Processo recebe os parâmetros identificador, host e port e os atribui aos respectivos atributos da classe.

A classe Processo possui métodos getter e setter para cada um dos atributos (getIdentificador(), getHost(), getPort(), setIdentificador(), setHost(), setPort()), que permitem obter e modificar os valores desses atributos.

A Classe “Processos”:





O código fornecido representa uma classe chamada Processos que implementa o padrão Singleton e gerencia os processos em um sistema distribuído.

A classe Processos possui um atributo estático chamado uniqueInstance, que representa a única instância da classe. Esse atributo é inicializado como nulo.

A classe possui vários atributos, como processos (um mapa que mapeia identificadores de processos para objetos da classe Processo), me (o objeto Processo que representa o próprio processo), lider (o objeto Processo que representa o líder do sistema), e totalProcesso (o número total de processos no sistema).

A classe possui métodos getter e setter para cada um dos atributos, permitindo obter e modificar os valores desses atributos.

O construtor da classe Processos é privado, impedindo que outras classes instanciem diretamente objetos dessa classe.

A classe possui um método estático chamado getInstance() que implementa o padrão Singleton. Esse método retorna a única instância da classe Processos. Se a instância ainda não foi criada, o método cria uma nova instância e a atribui ao atributo uniqueInstance.

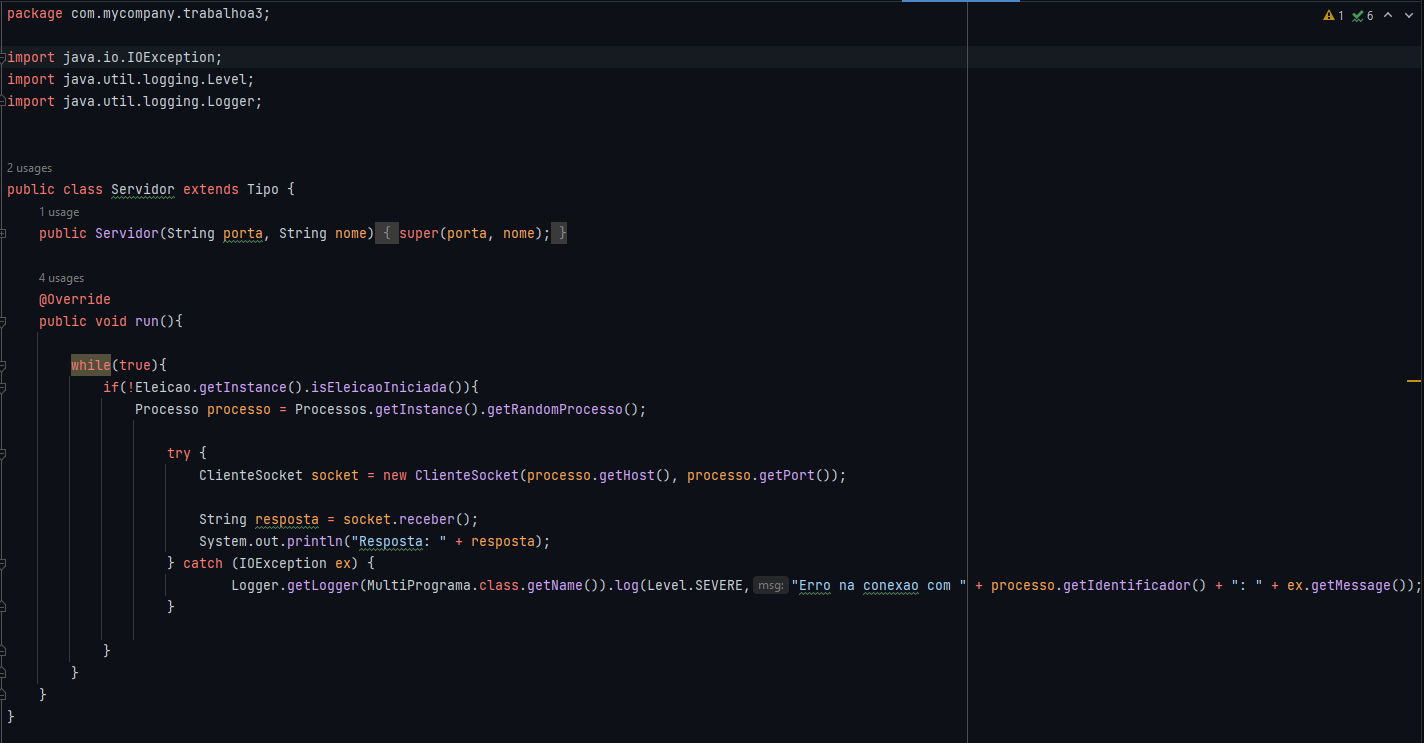
O método config() é responsável por configurar os atributos processos, me, lider e totalProcesso da classe Processos com os valores fornecidos como parâmetros.

O método getProcessoByIdentificador(Integer identificador) retorna o objeto Processo correspondente ao identificador fornecido.

O método getRandomProcesso() retorna um objeto Processo aleatório do mapa de processos.

O método checkServidor() verifica se o servidor (representado pelo objeto Processo líder) está disponível. Ele cria uma instância de ClienteSocket para se conectar ao líder e enviar uma mensagem de ping. Se não houver exceção de IOException, significa que o líder está disponível e o método retorna true. Caso contrário, retorna false.

A classe “Servidor”:



O código fornecido representa uma classe chamada Servidor que estende a classe abstrata Tipo.

A classe Servidor possui um construtor que recebe dois parâmetros: porta (a porta em que o servidor será executado) e nome (o nome do servidor).

A classe implementa o método run() que é chamado quando a thread do servidor é iniciada. Dentro desse método, há um loop infinito while(true) que aguarda a recepção de mensagens de outros processos.

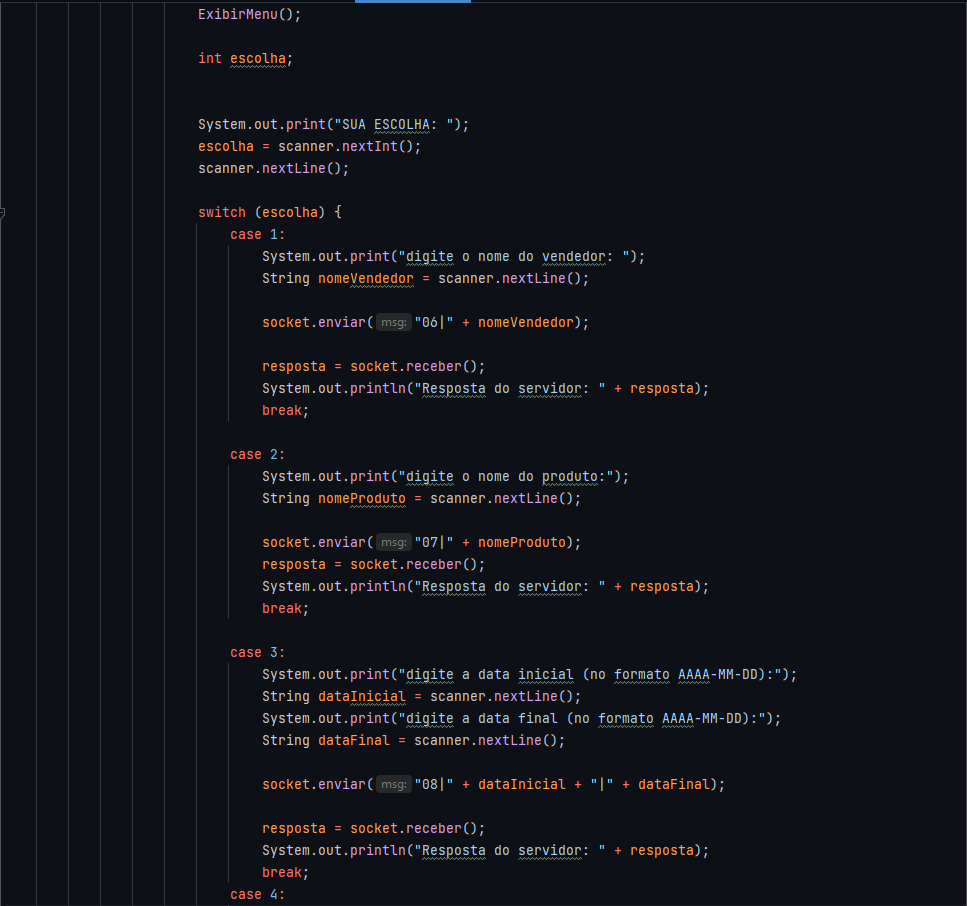
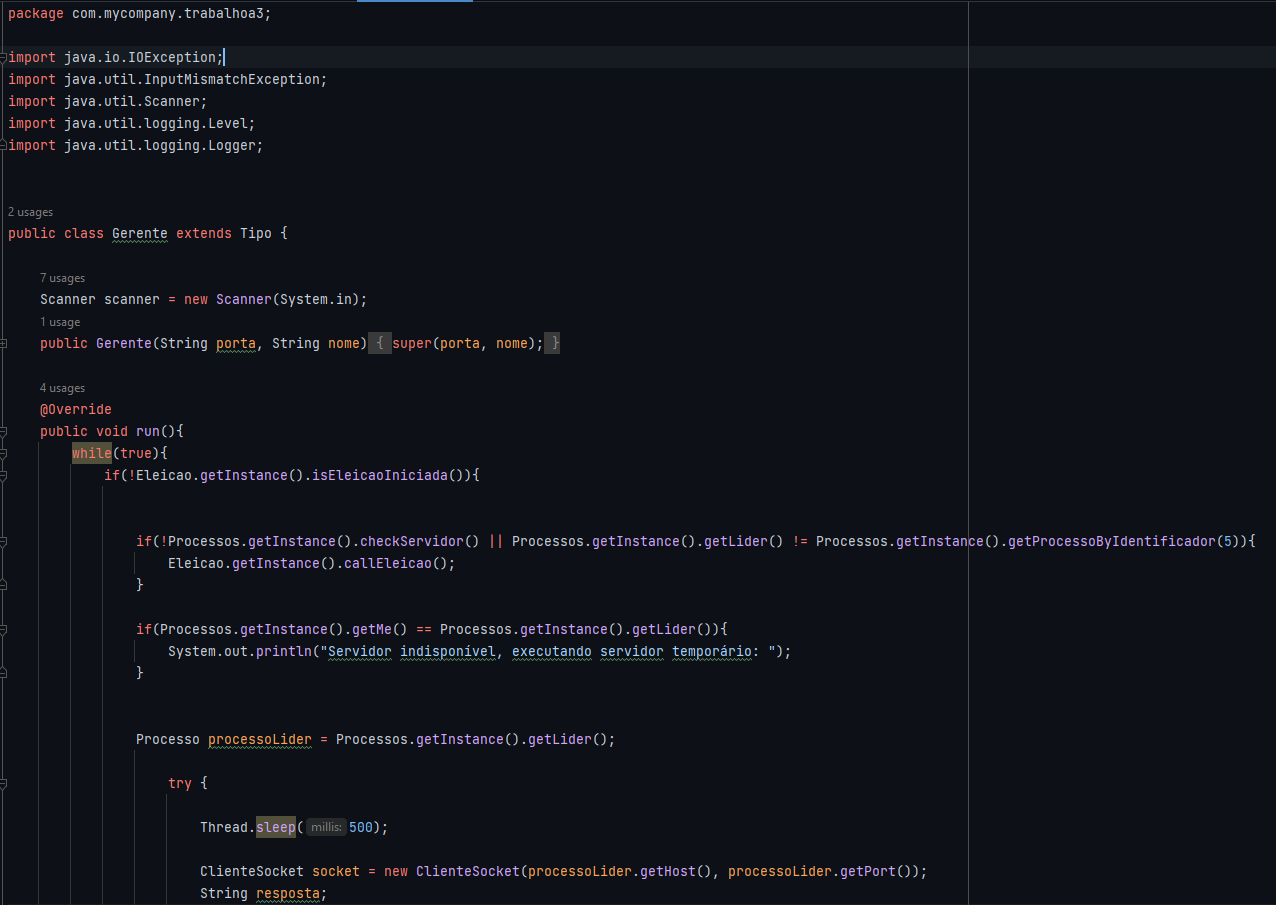
Antes de iniciar uma eleição (verificado pelo método isEleicaoIniciada() da classe Eleicao), o servidor seleciona um processo aleatório usando o método getRandomProcesso() da classe Processos.

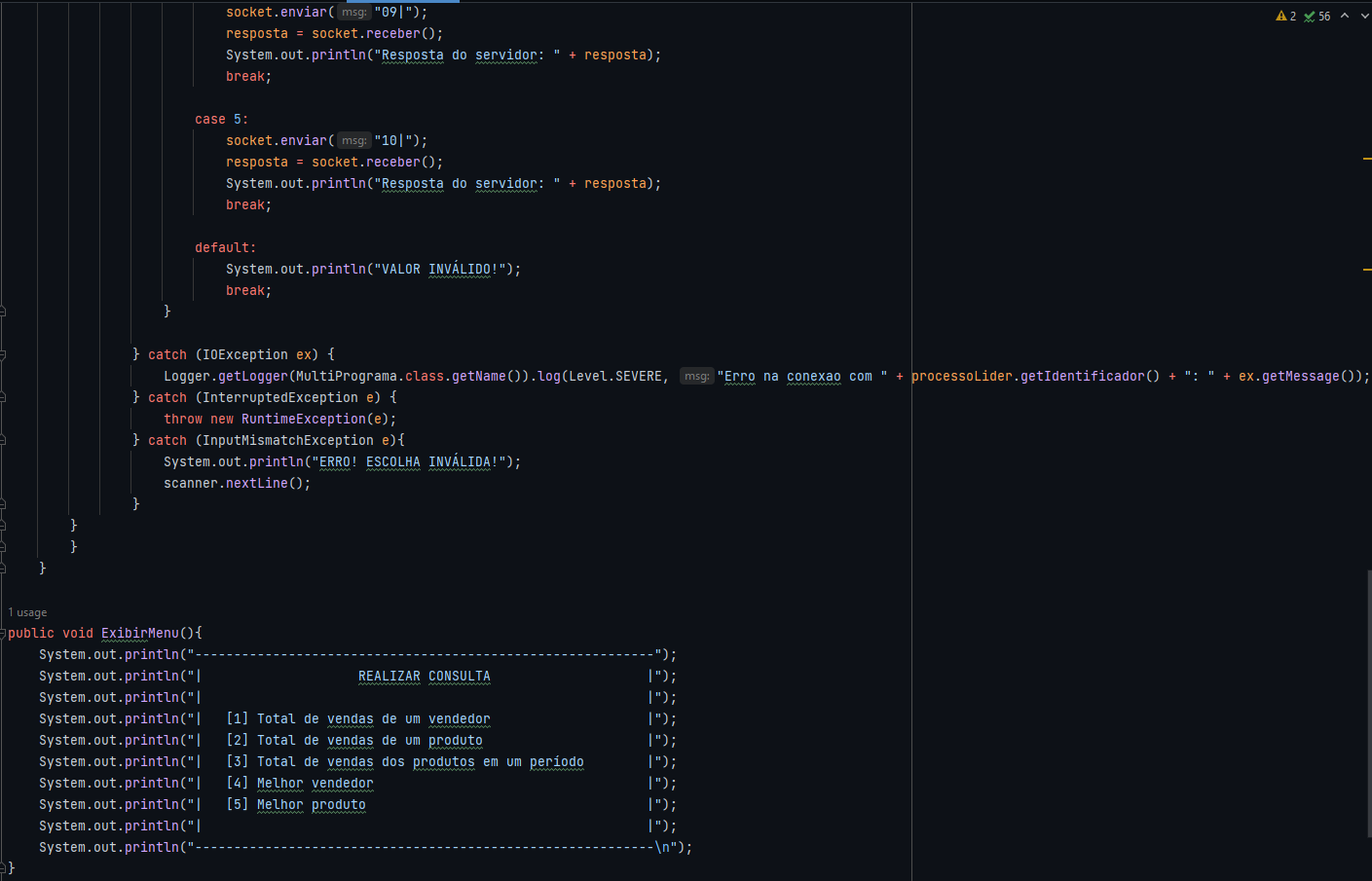
Em seguida, é criada uma instância de ClienteSocket para se conectar ao processo selecionado e receber uma mensagem.

A mensagem recebida de clientes é armazenada na variável resposta e é exibida no console usando System.out.println().

Se ocorrer uma exceção de IOException ao tentar se conectar ao processo ou receber a mensagem, um registro de log é feito usando a classe Logger.

A classe “Gerente”:





O código fornecido representa uma classe chamada Gerente que estende a classe abstrata Tipo. Essa classe representa um gerente em um sistema distribuído e implementa a lógica para interagir com o servidor e realizar consultas.

A classe Gerente possui um construtor que recebe dois parâmetros: porta (a porta em que o gerente será executado) e nome (o nome do gerente).

A classe implementa o método run() que é chamado quando a thread do gerente é iniciada. Dentro desse método, há um loop infinito while(true) que executa a lógica de interação com o servidor.

Antes de realizar qualquer ação, é verificado se uma eleição já foi iniciada usando o método isEleicaoIniciada() da classe Eleicao. Se não houver eleição em andamento, o código continua.

Em seguida, é verificado se o servidor está indisponível (usando o método checkServidor() da classe Processos) ou se o líder atual não é o processo atual do gerente. Se alguma dessas condições for verdadeira, uma nova eleição é iniciada usando o método callEleicao() da classe Eleicao.

Se o processo atual do gerente for o líder, ele exibe um menu no console usando o método ExibirMenu() e aguarda a escolha do usuário.

O usuário digita sua escolha e essa escolha é armazenada na variável “escolha”.

Em seguida, um switch-case é usado para executar a ação correspondente à escolha do usuário:

Se a escolha for 1, o usuário é solicitado a digitar o nome do vendedor e essa informação é enviada ao servidor usando socket.enviar(). Em seguida, a resposta do servidor é recebida usando socket.receber() e exibida no console.

Se a escolha for 2, o usuário é solicitado a digitar o nome do produto e essa informação é enviada ao servidor. A resposta do servidor é recebida e exibida.

Se a escolha for 3, o usuário é solicitado a digitar a data inicial e a data final (no formato AAAA-MM-DD). Essas informações são enviadas ao servidor e a resposta é recebida e exibida.

Se a escolha for 4, uma mensagem é enviada ao servidor para obter o melhor vendedor. A resposta do servidor é recebida e exibida.

Se a escolha for 5, uma mensagem é enviada ao servidor para obter o melhor produto. A resposta do servidor é recebida e exibida.

Se a escolha não corresponder a nenhum dos casos anteriores, uma mensagem de valor inválido é exibida.

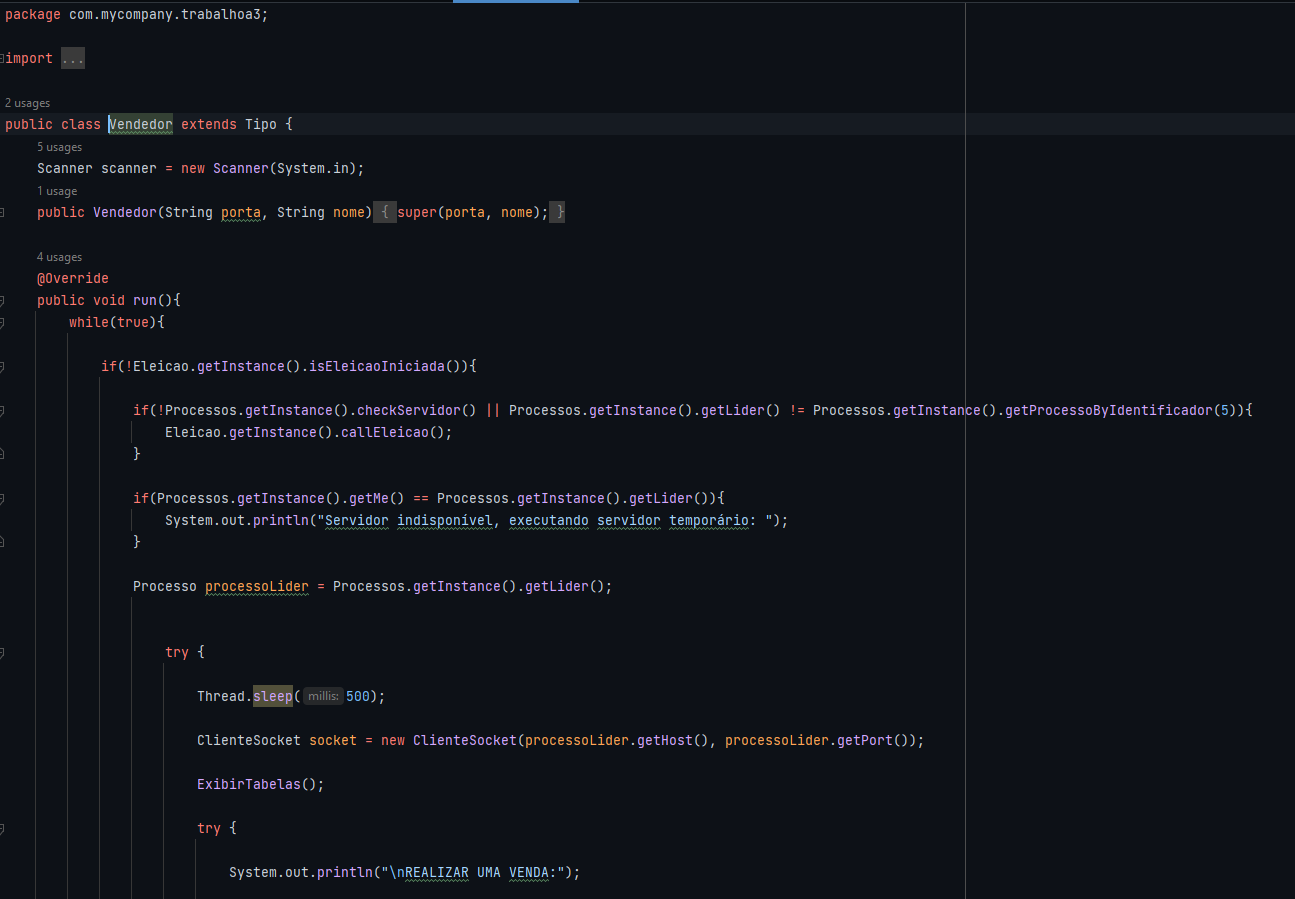
Se ocorrer uma exceção de IOException ao tentar se conectar ao líder ou receber a resposta do servidor, um registro de log é feito usando a classe Logger.

Se ocorrer uma exceção de InterruptedException, é lançada uma exceção RuntimeException.

Se ocorrer uma exceção de InputMismatchException, uma mensagem de erro é exibida e o programa espera pela próxima entrada do usuário.

O método ExibirMenu() é responsável por exibir o menu de opções disponíveis para o usuário.

A classe “Vendedor”:







O código fornecido representa uma classe chamada Vendedor que estende a classe abstrata Tipo. Essa classe representa um vendedor em um sistema distribuído e implementa a lógica para realizar vendas e exibir as tabelas de vendedores e produtos.

A classe Vendedor possui um construtor que recebe dois parâmetros: porta (a porta em que o vendedor será executado) e nome (o nome do vendedor).

A classe implementa o método run() que é chamado quando a thread do vendedor é iniciada. Dentro desse método, há um loop infinito while(true) que executa a lógica de interação com o servidor.

Antes de realizar qualquer ação, é verificado se uma eleição já foi iniciada usando o método isEleicaoIniciada() da classe Eleicao. Se não houver eleição em andamento, o código continua.

Em seguida, é verificado se o servidor está indisponível (usando o método checkServidor() da classe Processos) ou se o líder atual não é o processo atual do vendedor. Se alguma dessas condições for verdadeira, uma nova eleição é iniciada usando o método callEleicao() da classe Eleicao.

Se o processo atual do vendedor for o líder, ele exibe as tabelas de vendedores e produtos usando o método ExibirTabelas().

Em seguida, o vendedor solicita informações ao usuário para realizar uma venda. O nome do vendedor, o nome do produto, a quantidade vendida e a data da venda são solicitados.

Os dados da venda são enviados ao servidor usando socket.enviar(). Se ocorrer uma exceção de InputMismatchException, uma mensagem de erro é exibida e o método run() é chamado novamente para solicitar novos dados ao usuário.

A resposta do servidor é recebida usando socket.receber() e exibida no console.

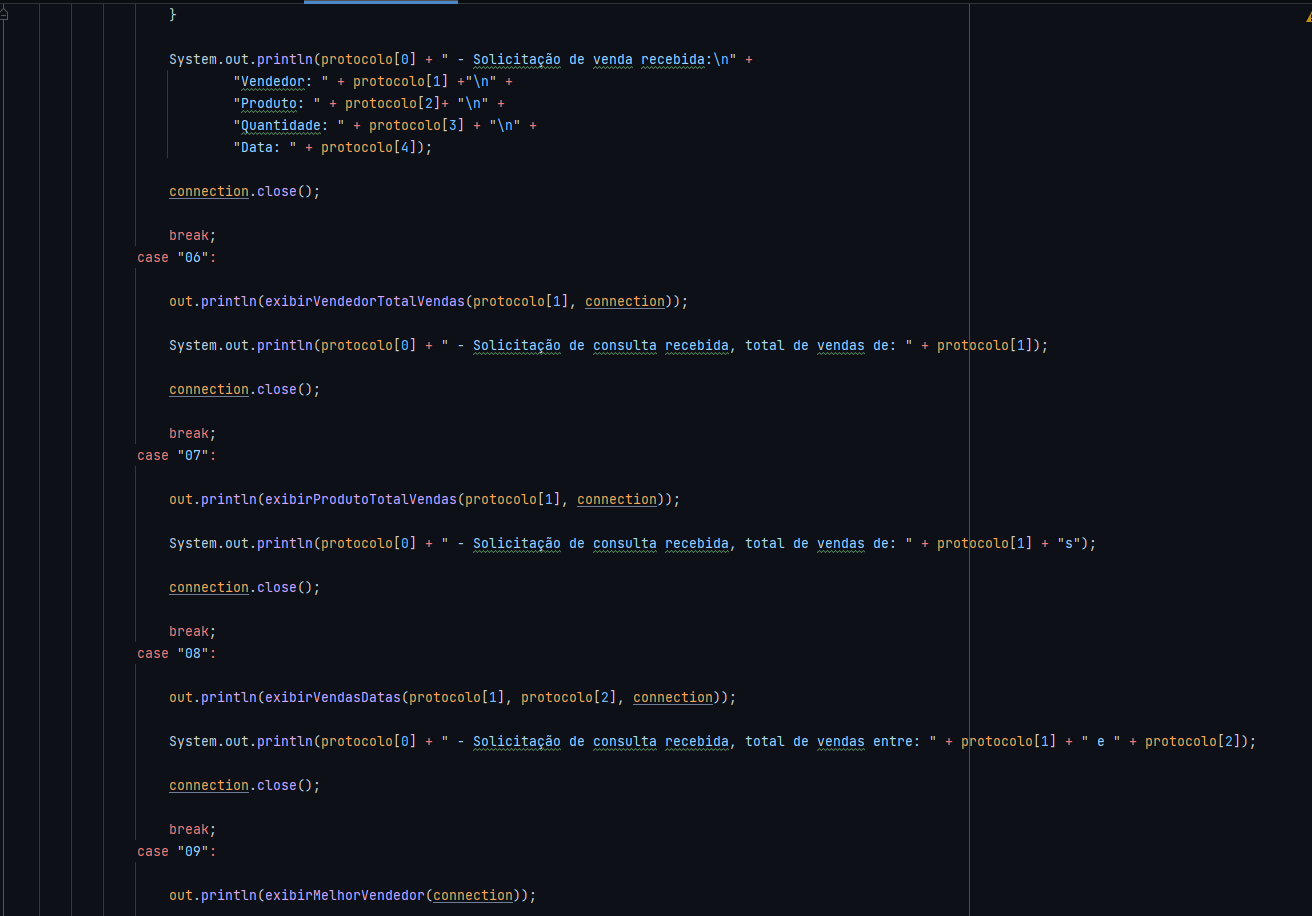
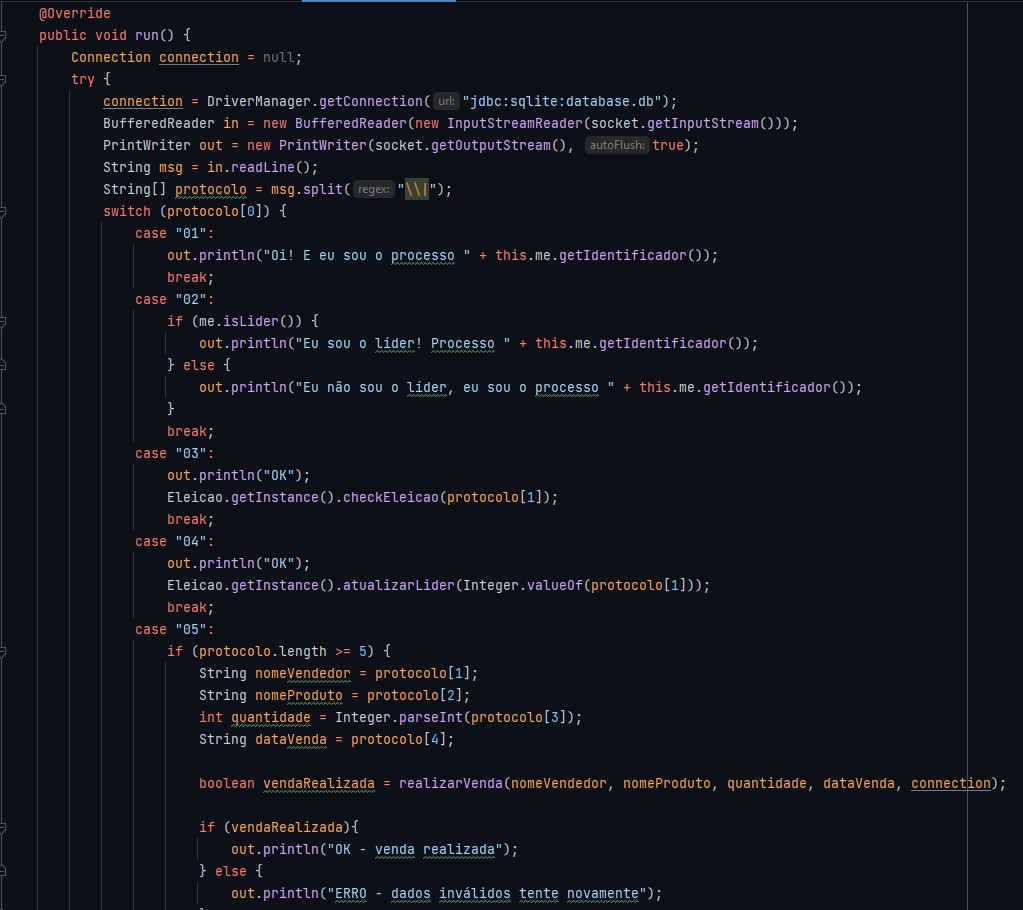
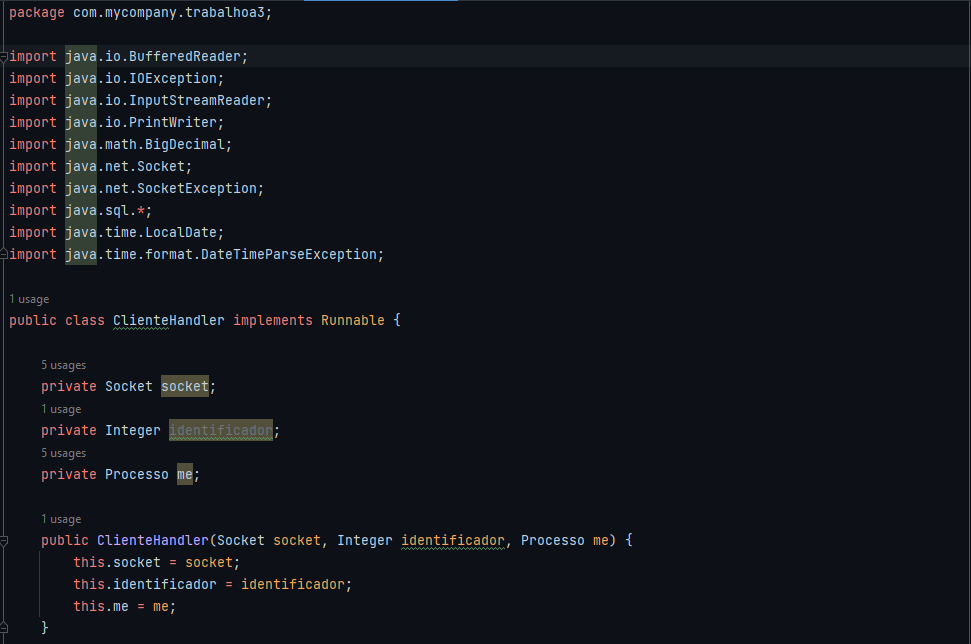
Após realizar a venda, a thread é pausada por 5 segundos utilizando o Thread.sleep().

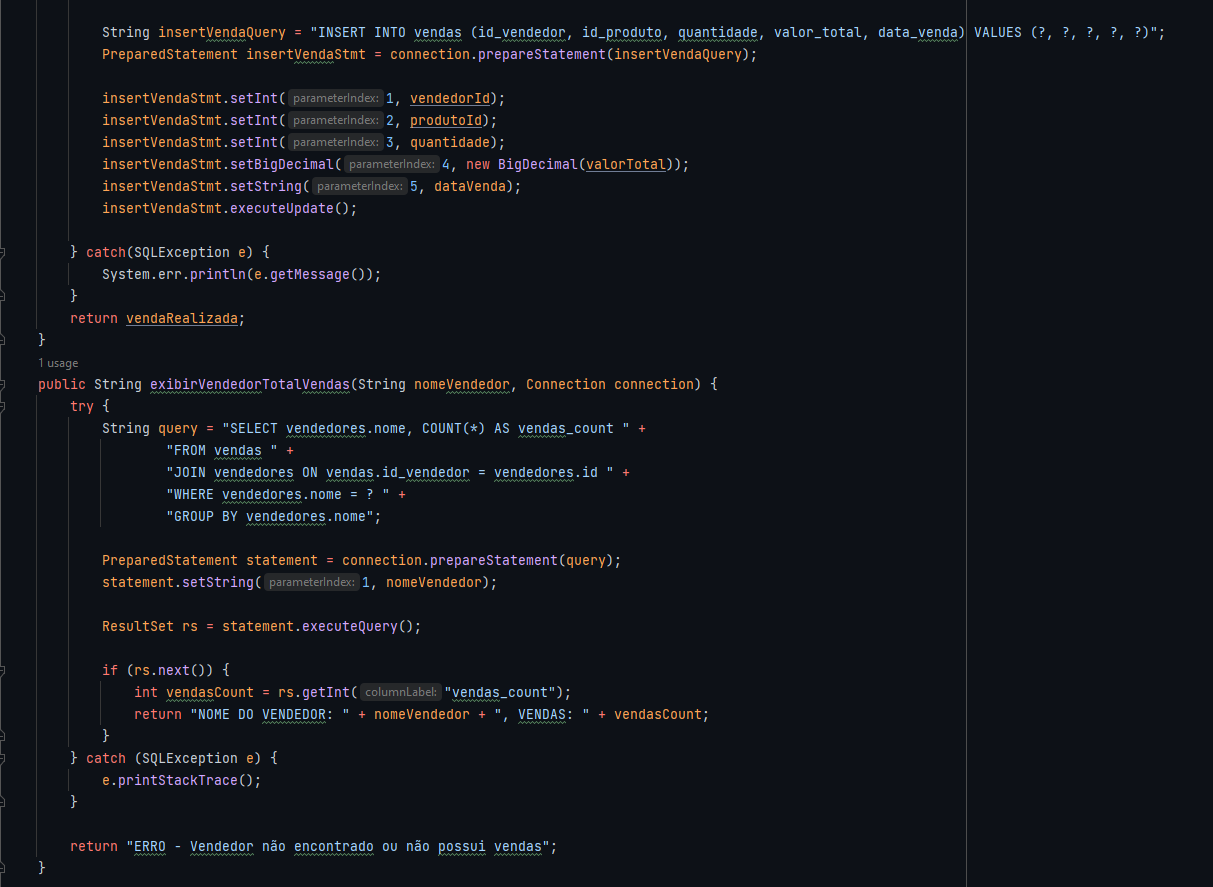
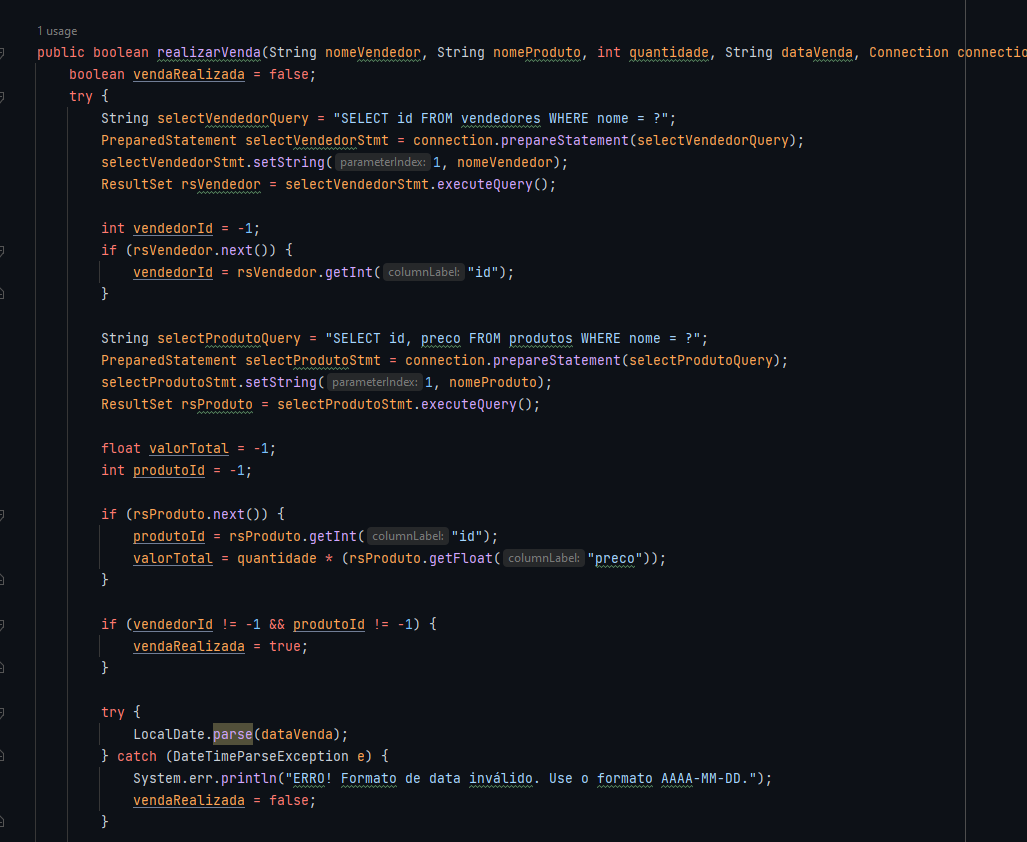
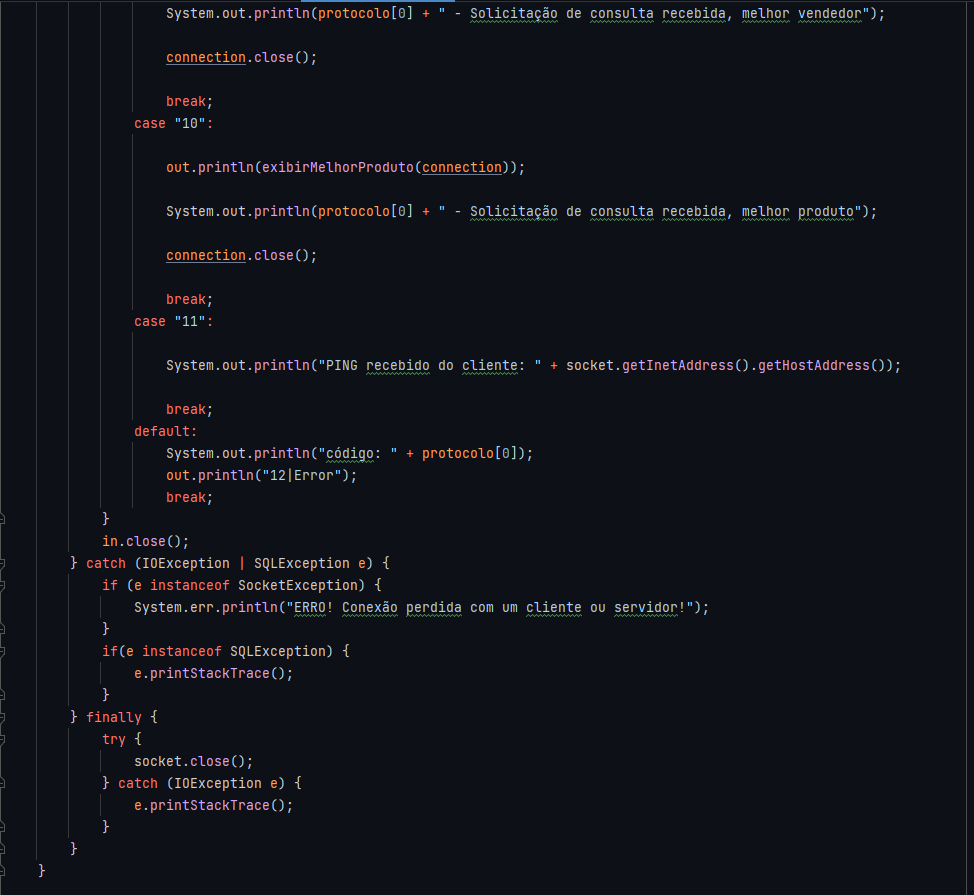
Se ocorrer uma exceção de IOException ao tentar se conectar ao líder ou receber a resposta do servidor, um registro de log é feito usando a classe Logger.

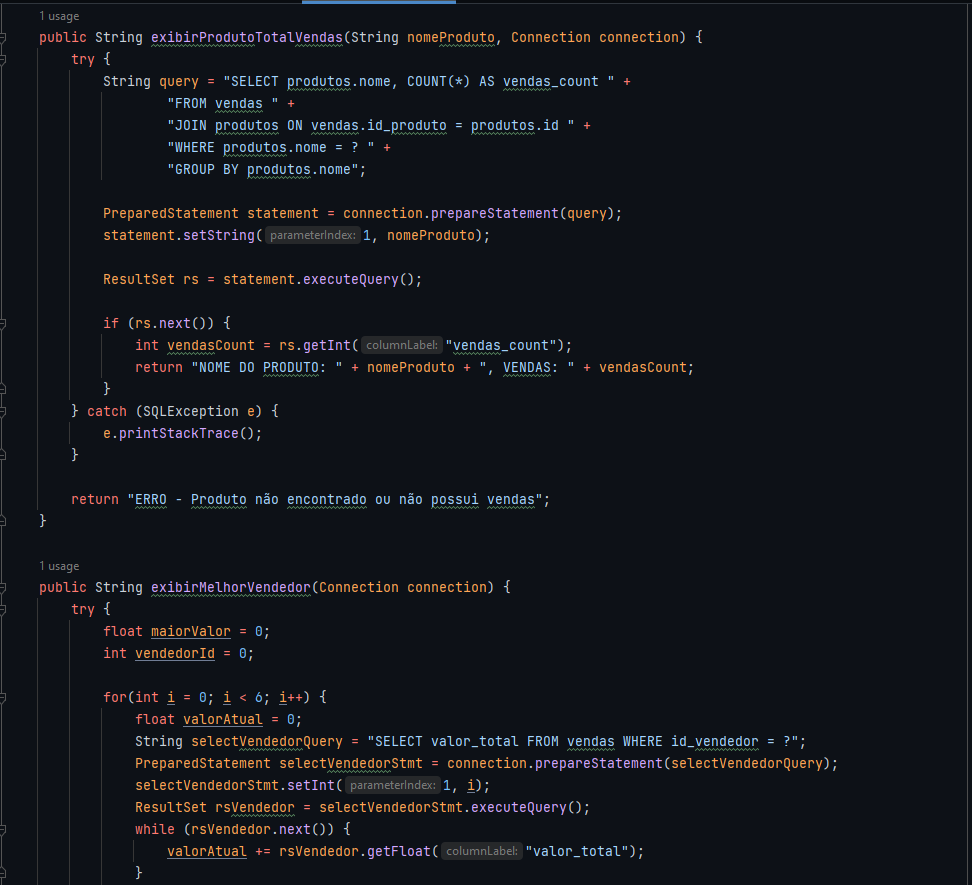
Se ocorrer uma exceção de InterruptedException, é lançada uma exceção RuntimeException.

O método ExibirTabelas() é responsável por se conectar ao banco de dados SQLite (database.db) e recuperar os registros das tabelas de vendedores e produtos. Em seguida, ele exibe os nomes dos vendedores e produtos no console.

A classe “ClienteHandler”:









O código fornecido é uma implementação de um manipulador de cliente em um servidor. Ele lida com as solicitações recebidas dos clientes e executa ações com base nos protocolos definidos. Vou explicar algumas partes-chave do código:

A classe ClienteHandler implementa a interface Runnable, o que significa que ela pode ser executada em uma thread separada.

O construtor ClienteHandler recebe o soquete de conexão, o identificador do cliente e uma referência ao objeto Processo. O objeto Processo parece ser uma classe externa relacionada ao controle de processos do servidor.

O método run é onde a lógica principal do manipulador de cliente está implementada. Ele é executado quando a thread do manipulador de cliente é iniciada.

Dentro do método run, uma conexão com o banco de dados é estabelecida usando o driver JDBC para SQLite.

Em seguida, os objetos BufferedReader e PrintWriter são criados a partir dos fluxos de entrada e saída do soquete, permitindo a comunicação com o cliente.

A mensagem recebida do cliente é lida usando in.readLine(). A mensagem é dividida com base no caractere '|' para separar o código do protocolo e os parâmetros.

Um switch-case é usado para determinar a ação a ser executada com base no código do protocolo recebido.

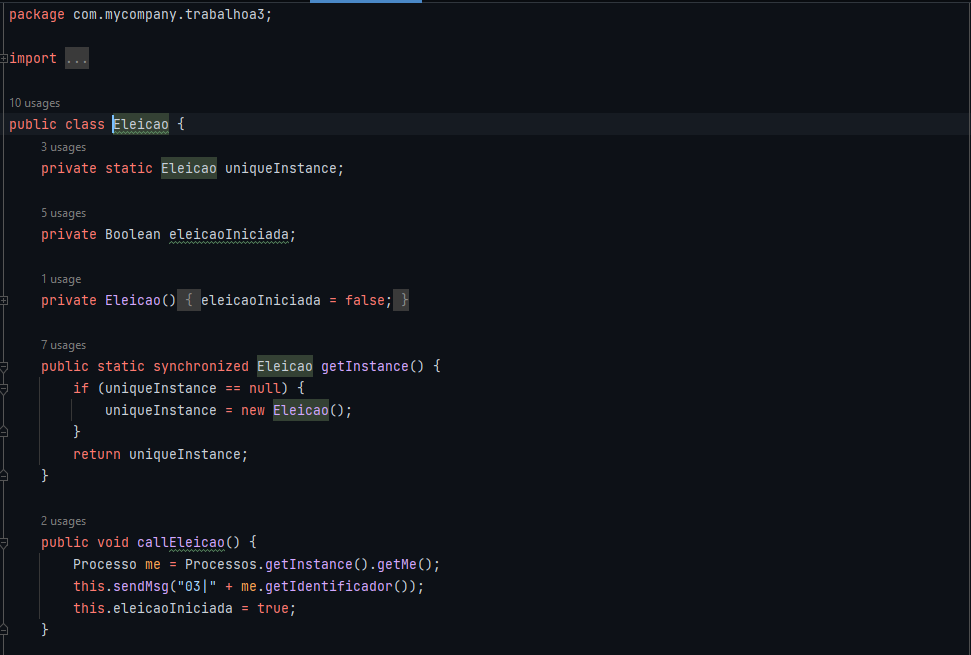
Cada caso no switch-case executa uma ação específica com base no protocolo.

Existem várias consultas SQL implementadas no código, como exibirVendedorTotalVendas, exibirProdutoTotalVendas, exibirMelhorVendedor, exibirMelhorProduto e exibirVendasDatas. Essas consultas são usadas para recuperar informações do banco de dados e retornar as respostas aos clientes.

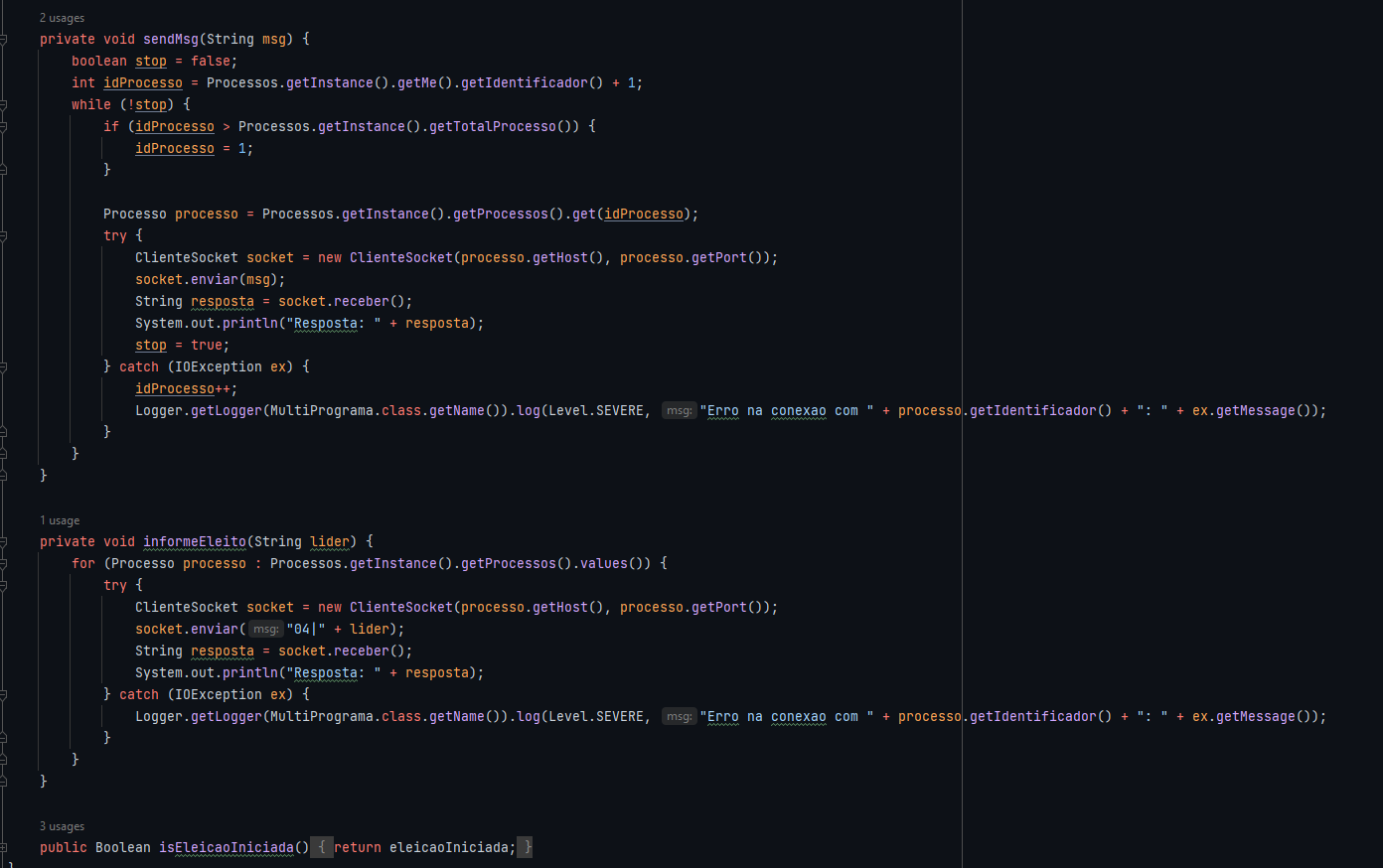
No final de cada caso, a conexão com o banco de dados é fechada usando connection.close().

O código trata exceções, como IOException e SQLException, e também lida com possíveis erros de conexão com o cliente ou servidor.

A classe “Eleicao”:







A classe Eleicao é responsável por representar um processo de eleição e contém métodos para iniciar e gerenciar a eleição. Vamos analisar os principais componentes e métodos da classe:

uniqueInstance: É uma variável estática que armazena a única instância da classe Eleicao. Ela utiliza o padrão de projeto Singleton para garantir que apenas uma instância da classe possa ser criada.

eleicaoIniciada: É uma variável booleana que indica se a eleição está em andamento ou não.

Construtor Eleicao(): É um construtor privado que é chamado apenas internamente na classe. Ele inicializa a variável eleicaoIniciada como false.

getInstance(): É um método estático que implementa o padrão Singleton. Ele retorna a única instância da classe Eleicao. Se a instância ainda não foi criada, o método cria uma nova instância e a armazena na variável uniqueInstance. Caso contrário, ele retorna a instância existente.

callEleicao(): Este método é responsável por iniciar a eleição. Ele obtém a referência do processo atual através do método getMe() da classe Processos e envia uma mensagem para os outros processos informando que a eleição foi iniciada. Ele também define a variável eleicaoIniciada como true.

checkEleicao(String msg): Este método é chamado quando uma mensagem de verificação da eleição é recebida. Ele verifica se o processo atual está na lista de candidatos recebida na mensagem. Se não estiver, o processo se adiciona à lista e envia uma mensagem atualizada para os outros processos. Caso contrário, a eleição é considerada finalizada, o líder é determinado com base no identificador máximo na lista de candidatos e o método informeEleito(lider) é chamado para informar a todos os processos sobre o líder eleito. Em seguida, o método atualizarLider(Integer lider) é chamado para atualizar o estado do líder no objeto Processo correspondente.

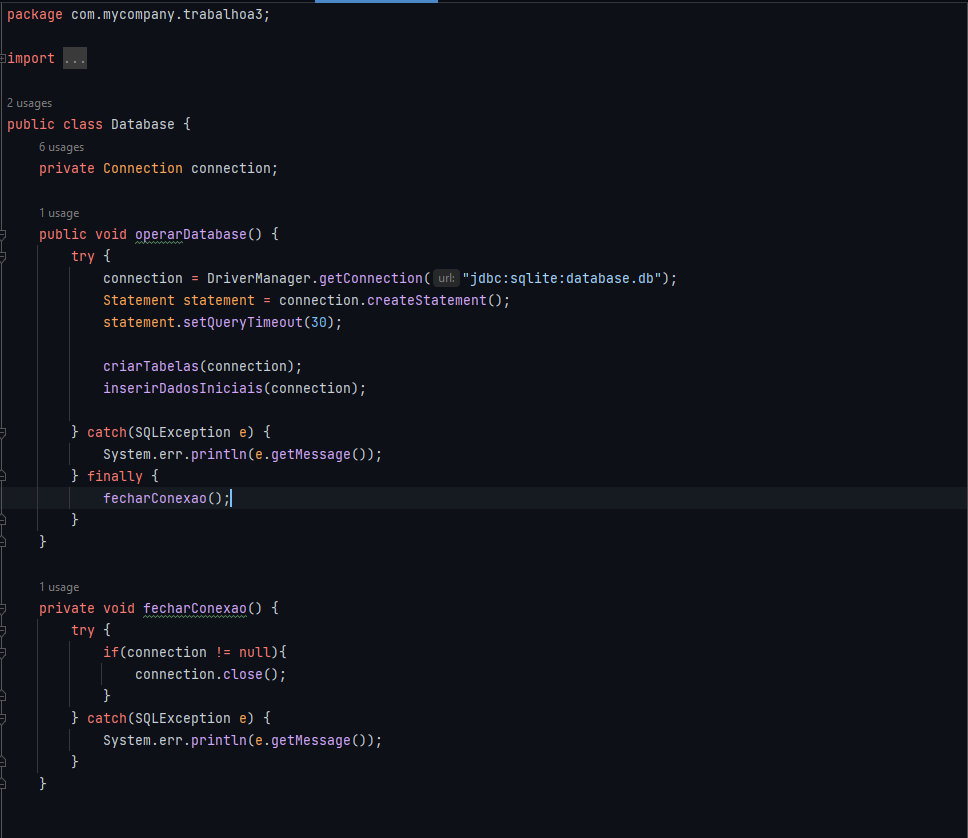
atualizarLider(Integer lider): Este método é responsável por atualizar o estado do líder no objeto Processo correspondente ao processo identificado pelo parâmetro lider. Ele marca o processo como líder através do método setIsLider(true) e atualiza o líder no objeto Processos através do método setLider(processo). Por fim, define a variável eleicaoIniciada como false.

sendMsg(String msg): Este método é usado para enviar uma mensagem para outro processo. Ele itera pelos processos existentes, começando pelo próximo processo após o processo atual. Cria um objeto ClienteSocket para estabelecer a conexão com o processo destino e envia a mensagem. Em seguida, aguarda e imprime a resposta recebida. Se ocorrer uma exceção de E/S (entrada/saída), a conexão com o processo é considerada falha e o próximo processo é tentado.

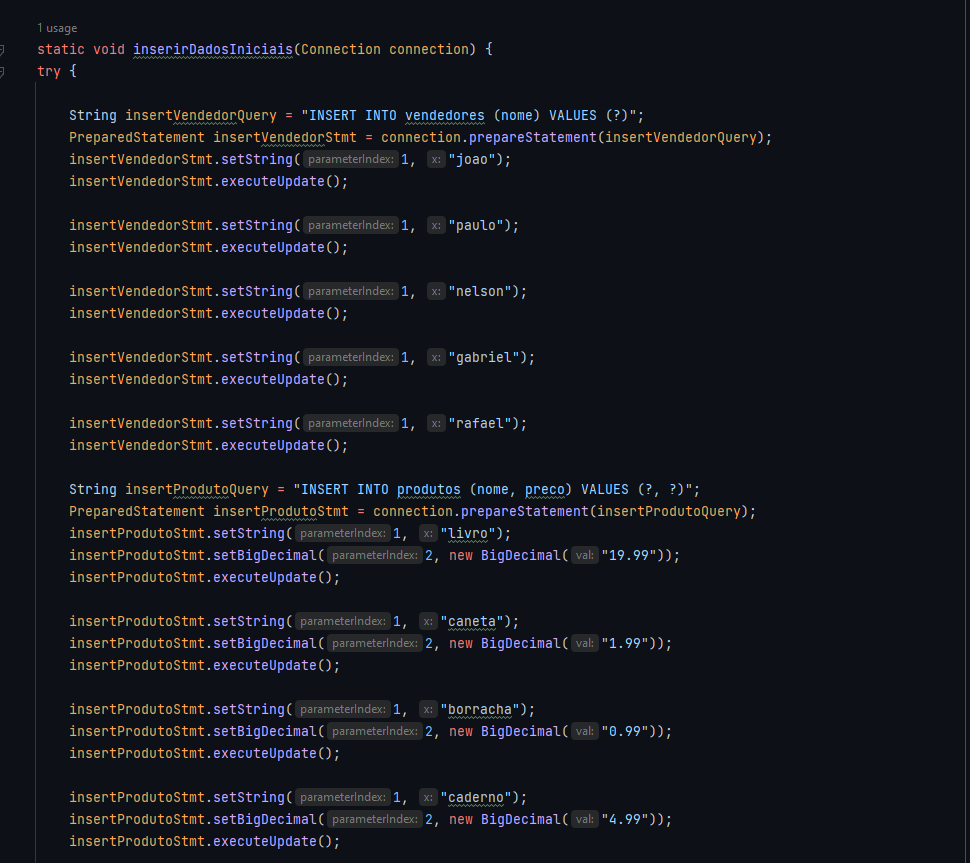
informeEleito(String lider): Este método envia uma mensagem para todos os processos informando o líder eleito. Itera pelos processos existentes e cria um objeto ClienteSocket para estabelecer a conexão com cada processo. Envia a mensagem contendo o líder eleito e aguarda a resposta. Em seguida, imprime a resposta recebida. Se ocorrer uma exceção de E/S, a conexão com o processo é considerada falha.

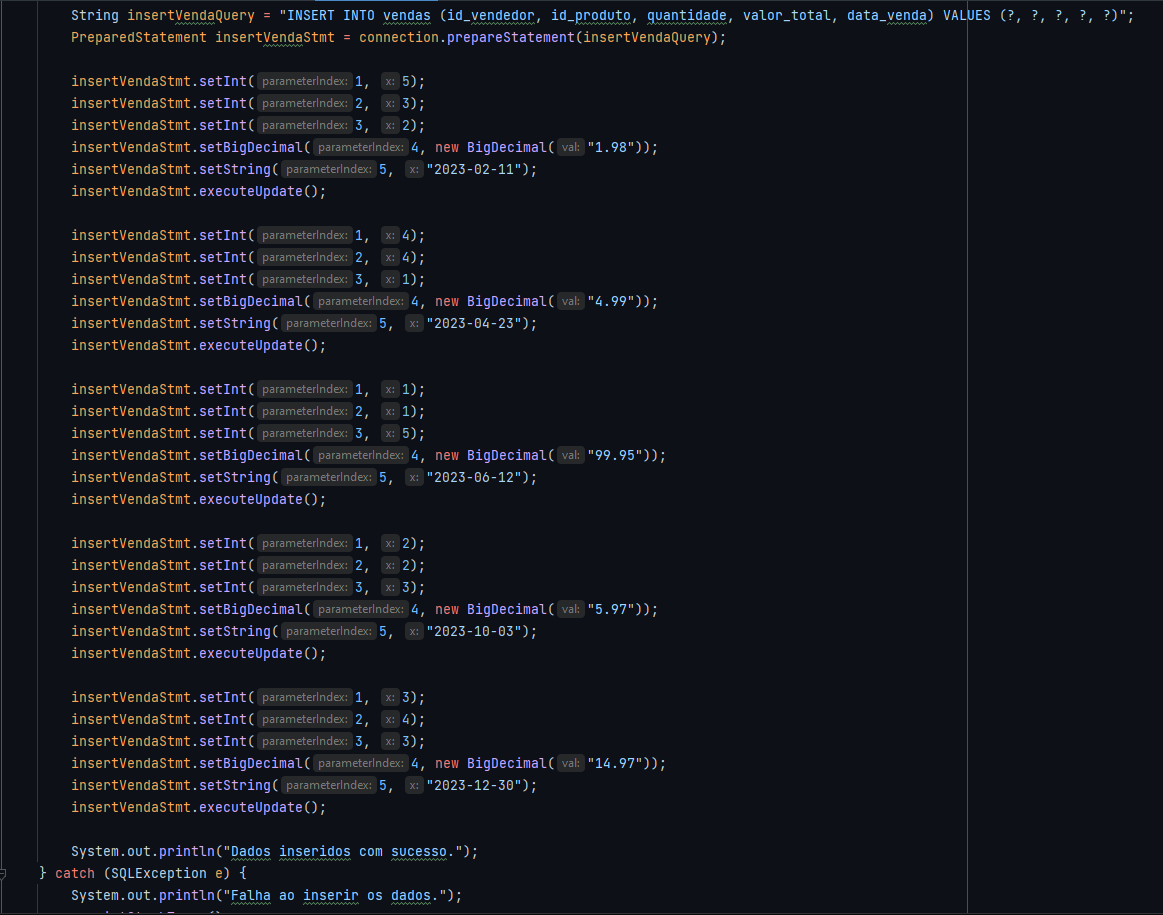
isEleicaoIniciada(): Este método retorna o valor da variável eleicaoIniciada, indicando se a eleição está em andamento ou não.

A classe “Database”:









A classe Database é responsável por realizar operações relacionadas ao banco de dados, como criar tabelas e inserir dados iniciais. Vamos analisar os principais componentes e métodos da classe:

connection: É um objeto do tipo Connection que representa a conexão com o banco de dados.

operarDatabase(): É o método principal da classe que realiza as operações no banco de dados. Ele tenta estabelecer a conexão com o banco de dados, cria um objeto Statement para executar as consultas SQL e chama os métodos criarTabelas() e inserirDadosIniciais(). Em caso de exceção SQLException, a mensagem de erro é impressa. Independentemente do resultado, o método fecharConexao() é chamado para fechar a conexão com o banco de dados.

fecharConexao(): É um método privado que fecha a conexão com o banco de dados. Ele verifica se a conexão não é nula antes de fechá-la e trata qualquer exceção SQLException que possa ocorrer durante o fechamento da conexão.

criarTabelas(Connection connection): É um método estático que recebe a conexão com o banco de dados como parâmetro e é responsável por criar as tabelas necessárias no banco de dados. Ele utiliza um objeto Statement para executar as instruções SQL de criação das tabelas. Se ocorrer uma exceção SQLException, uma mensagem de falha é exibida e a pilha de exceção é impressa.

inserirDadosIniciais(Connection connection): É um método estático que recebe a conexão com o banco de dados como parâmetro e é responsável por inserir os dados iniciais nas tabelas do banco de dados. Ele utiliza objetos PreparedStatement para criar consultas parametrizadas e inserir os valores correspondentes. Os dados iniciais incluem vendedores, produtos e vendas. Se ocorrer uma exceção SQLException, uma mensagem de falha é exibida e a pilha de exceção é impressa.

**3. ALGORITMO DE ELEIÇÃO UTILIZADO**

O algoritmo utilizado para a eleição de um novo líder foi o Bully. O algoritmo de Bully serve para eleger um líder entre processos identificados por um identificador numérico, único, fixo e atribuído antes do início da eleição. E cada um dos processos pode se comunicar com qualquer outro no sistema. A execução do algoritmo busca eleger o processo de maior identificador e fazer com que todos reconheçam o novo líder.

Caso um dos processos identifique a perda de contato com o líder, inicia uma nova eleição enviando a todos os outros uma mensagem contendo seu identificador. Então todos os nós respondem ao processo que iniciou a eleição com os seus próprios identificadores. Se o processo que iniciou a eleição verifica possuir o maior identificador entre todos os outros, proclama-se líder e avisa todos os outros. Se não aguarda que o processo de maior identificador inicie uma eleição e se torne líder.

O líder vai assumir temporariamente como servidor, até que o servidor original volte a ficar online. Quando o servidor sobe de novo, os clientes identificam que o mesmo está online, e então chamam a nova eleição, que por sua vez vai eleger ele como líder novamente.

**4. CONCLUSÃO E INSTRUÇÕES**

Para utilizar o aplicativo você precisa ter a versão mais recente do Java SE Development Kit (JDK) instalada em seu computador e utilizar o sistema operacional Windows.

1. Baixe o arquivo .zip do repositório e descompacte.
2. Caso o arquivo “database.db” não exista, execute “startServidor.bat” uma vez. Isso irá abrir o processo “Servidor” e criar o banco de dados com os dados necessários.
3. Com o arquivo “database.db” disponível na pasta, execute “startProcessos.bat”. Isso irá abrir 5 processos diferentes, um processo “Servidor”, dois processos “Gerente” e dois processos “Vendedor”.
4. Feito isso o aplicativo estará pronto para o uso.
5. Para testar o comportamento caso o servidor seja derrubado, feche o processo do servidor (o processo com o identificador 5).
6. Agora, ao tentar realizar alguma solicitação, a eleição será chamada e o processo com maior identificador assumirá as funcionalidades do servidor (líder).
7. Para fazer com que o servidor retorne e retome suas funcionalidades, execute o arquivo “startServidor.bat” e realize alguma solicitação com algum dos clientes.
8. Feito isso uma eleição será chamada e o servidor se tornará líder novamente e retomará suas funções (já que tem o maior identificador).