SISTEMA DE ESTOQUE

O presente relatório descreve o projeto desenvolvido em Java com Maven, utilizando SQL e Hibernate, para um sistema de controle de estoque. O sistema é projetado para auxiliar no gerenciamento e controle das operações relacionadas a compras, estoques, fornecedores, produtos e usuários.

O projeto é composto por 8 classes principais, que são: Compra, Comprador, Comprador Físico, Comprador Jurídico, Estoque, Fornecedor, Produto e Usuário. Essas classes representam as entidades centrais do sistema e desempenham um papel fundamental no armazenamento e manipulação dos dados.

A classe "**Usuário**" é uma entidade persistente que representa um usuário no sistema. Ela possui atributos como o id do usuário, nome de usuário, senha e senha criptografada.

A classe implementa métodos para acessar e modificar esses atributos.
A senha é armazenada criptografada usando o algoritmo BCrypt, o que ajuda a proteger as informações de login dos usuários. A classe fornece um método para atualizar a senha, que também realiza a criptografia usando o BCrypt.

A classe "**Produto**" representa um produto no sistema. Ela possui atributos como id do produto, preço e nome. A classe também possui relacionamentos com outras entidades, como "Estoque", "Fornecedor" e "Compra".

A classe possui um relacionamento um-para-um com a entidade "Estoque", onde um produto está associado a um estoque específico. O relacionamento é bidirecional, com a propriedade mappedBy indicando que o mapeamento é feito através do atributo "produto" na entidade "Estoque".

Também existe um relacionamento muitos-para-um com a entidade "Fornecedor", indicando que um produto está associado a um fornecedor específico. O relacionamento é mapeado através do atributo "idFornecedor" na tabela do banco de dados.

Além disso, a classe possui um relacionamento muitos-para-muitos com a entidade "Compra", representando as vendas em que o produto está envolvido. O relacionamento é mapeado através do atributo "produtos" na entidade "Compra".

A classe fornece métodos para acessar e modificar os atributos, além de métodos para obter informações relacionadas, como a quantidade em estoque e o CNPJ do fornecedor.

A classe "**Estoque**" representa o estoque de um produto no sistema. Ela possui atributos como id do estoque e quantidade disponível. A classe também possui um relacionamento com a entidade "Produto".

O atributo idEstoque é a chave primária da entidade, e seu valor é gerado automaticamente pelo banco de dados. O atributo quantidade armazena a quantidade disponível do produto em estoque.

A classe possui um relacionamento um-para-um com a entidade "Produto", onde um estoque está associado a um produto específico. O relacionamento é mapeado através do atributo "idProduto" na tabela do banco de dados.

A classe "**Fornecedor"** representa um fornecedor no sistema. Ela possui atributos como id do fornecedor, CNPJ e nome.

O atributo idFornecedor é a chave primária da entidade, e seu valor é gerado automaticamente pelo banco de dados. O atributo CNPJ armazena o CNPJ do fornecedor e o atributo Nome armazena o nome do fornecedor.

A classe possui um relacionamento um-para-muitos com a entidade "Produto". Um fornecedor pode fornecer vários produtos. O relacionamento é mapeado através do atributo "fornecedor" na entidade "Produto".

A classe fornece métodos para acessar e modificar os atributos, além de métodos para obter a lista de produtos fornecidos pelo fornecedor.

A classe "**Compra**" representa uma compra realizada no sistema. Ela possui atributos como id da compra e preço total.

A classe possui um relacionamento um-para-um com a entidade "Comprador". Cada compra está associada a um único comprador. O relacionamento é mapeado através do atributo "comprador" na entidade "Compra".

A classe possui um relacionamento muitos-para-muitos com a entidade "Produto". Uma compra pode envolver vários produtos, e cada produto pode estar presente em várias compras. O relacionamento é mapeado através do atributo "produtos" na entidade "Compra", utilizando uma tabela de junção "venda_produto".

A classe fornece métodos para acessar e modificar os atributos, além de métodos para obter a lista de produtos da compra.

A classe "Comprador" representa um comprador no sistema. Ela possui atributos como id do comprador e nome.

A classe possui um relacionamento muitos-para-muitos com a entidade "Compra". Um comprador pode realizar várias compras, e cada compra pode ter vários compradores. O relacionamento é mapeado através do atributo "vendas" na entidade "Comprador", utilizando uma tabela de junção.

A classe fornece métodos para acessar e modificar os atributos, além de um método para obter a lista de compras realizadas pelo comprador.

A classe "CompradorFisico" é uma subclasse de "Comprador" e representa um comprador pessoa física no sistema. Ela herda os atributos e métodos da classe "Comprador" e adiciona o atributo CPF.

A classe possui construtores para inicializar os atributos, métodos de acesso e modificação para o atributo CPF e um método toString() para retornar uma representação em string do comprador pessoa física.

A classe "CompradorJuridico" é uma subclasse de "Comprador" e representa um comprador pessoa jurídica no sistema. Ela herda os atributos e métodos da classe "Comprador" e adiciona o atributo CNPJ.

A classe possui construtores para inicializar os atributos, métodos de acesso e modificação para o atributo CNPJ e um método toString() para retornar uma representação em string do comprador pessoa jurídica.

Cada classe tem sua classe "DAO" é responsável por realizar operações de persistência relacionadas à cada entidade no banco de dados utilizando a JPA (Java Persistence API).Os métodos da classe permitem realizar operações como salvar uma compra no banco de dados, buscar uma compra pelo seu ID, buscar todas as compras registradas, e excluir uma compra do banco de dados.

Por sua vez, cada classe “DAO” possui uma interface "Inome_da_classeDAO", que define um contrato para classes que desejam implementar operações de acesso aos dados relacionados à cada entidade.

A classe App é responsável por proporcionar uma interface interativa e acessível para os usuários, além de implementar métodos de classe que validam entradas e coordenam as chamadas de métodos externos. Nela temos:

1. Método `criaUsuario(UsuarioDAO usuarioDAO)`: Recebe um objeto `UsuarioDAO` como parâmetro e permite criar um novo usuário. Solicita ao usuário a entrada de um nome de usuário e uma senha. Verifica se o nome de usuário já está em uso e, se não estiver, verifica se a senha possui pelo menos 6 caracteres. Se todas as condições forem atendidas, cria um novo objeto `Usuario` com o nome de usuário e a senha informados e o salva utilizando o `UsuarioDAO`.

2. Método `listaUsuario(UsuarioDAO usuarioDAO)`: Recebe um objeto `UsuarioDAO` como parâmetro e lista todos os usuários cadastrados. Utiliza o `UsuarioDAO` para buscar todos os usuários e exibe a lista resultante.

3. Método `buscaUsuario(UsuarioDAO usuarioDAO)`: Recebe um objeto `UsuarioDAO` como parâmetro e permite buscar um usuário pelo nome de usuário. Solicita ao usuário que informe o nome de usuário desejado e utiliza o `UsuarioDAO` para buscar o usuário correspondente. Se encontrado, exibe as informações do usuário. Caso contrário, exibe uma mensagem informando que nenhum usuário foi encontrado.

4. Método `verificaUsuario(UsuarioDAO usuarioDAO)`: Recebe um objeto `UsuarioDAO` como parâmetro e verifica a autenticação de um usuário. Solicita ao usuário que informe o nome de usuário e a senha. Utiliza o `UsuarioDAO` para buscar o usuário correspondente ao nome de usuário informado. Se encontrado, verifica se a senha informada corresponde à senha criptografada armazenada no objeto `Usuario`. Retorna `true` se a autenticação for bem-sucedida e `false` caso contrário.

5. Método `atualizaUsuario(UsuarioDAO usuarioDAO)`: Recebe um objeto `UsuarioDAO` como parâmetro e permite atualizar as informações de um usuário. Solicita

ao usuário que informe o nome de usuário e a senha para autenticação. Se a autenticação for bem-sucedida, solicita ao usuário o novo nome de usuário e a nova senha. Atualiza as informações do usuário correspondente utilizando o `UsuarioDAO`.

6. Método `deletaUsuario(UsuarioDAO usuarioDAO)`: Recebe um objeto `UsuarioDAO` como parâmetro e permite excluir um usuário. Solicita ao usuário que informe o nome de usuário e a senha para autenticação. Se a autenticação for bem-sucedida, utiliza o `UsuarioDAO` para excluir o usuário correspondente.
Aqui está uma descrição resumida dos métodos mencionados:

7. Método `criaComprador()`: Solicita ao usuário que informe o nome do comprador e se é uma pessoa física ou jurídica. Em seguida, solicita o CPF (se for pessoa física) ou o CNPJ (se for pessoa jurídica) e verifica se o número de caracteres é válido. Cria um objeto `CompradorFisico` ou `CompradorJuridico` e o salva utilizando o `compradorDAO`.

8. Método `criaProduto()`: Solicita ao usuário o número de produtos a serem criados. Para cada produto, solicita o nome e o preço, cria um objeto `Produto` com essas informações e o salva utilizando o `produtoDAO`. Em seguida, solicita a quantidade em estoque e cria um objeto `Estoque` com essa quantidade e o produto associado, salvando-o no `estoqueDAO`. Lista todos os fornecedores disponíveis, solicita o ID do fornecedor escolhido pelo usuário e associa-o ao produto. Atualiza as informações do produto utilizando o `produtoDAO`.

9. Método `validaCNPJ(cnpj)`: Verifica se o CNPJ informado possui pelo menos 14 caracteres. Retorna `true` se for válido, caso contrário, retorna `false`.

10. Método `validaCPF(cpf)`: Verifica se o CPF informado possui pelo menos 11 caracteres. Retorna `true` se for válido, caso contrário, retorna `false`.

11. Método `criaFornecedor(FornecedorDAO fornecedorDAO)`: Solicita ao usuário o número de fornecedores a serem criados. Para cada fornecedor, solicita o nome e o CNPJ, verificando se o CNPJ possui o número correto de caracteres. Verifica se o CNPJ já está cadastrado no sistema. Se não estiver duplicado, cria um objeto `Fornecedor` com as informações fornecidas e o salva utilizando o `fornecedorDAO`.

12. Método `listaFornecedores(FornecedorDAO fornecedorDAO)`: Lista todos os fornecedores cadastrados, utilizando o `fornecedorDAO`. Se não houver fornecedores cadastrados, exibe uma mensagem informando que ainda não existem fornecedores cadastrados.

13. Método `listaProdutos(ProdutoDAO produtoDAO)`: Lista todos os produtos cadastrados, utilizando o `produtoDAO`. Se não houver produtos cadastrados, exibe uma mensagem informando que ainda não foram cadastrados produtos.

14.. Método `listaCompradores(CompradorDAO compradorDAO)`: Lista todos os compradores cadastrados, utilizando o `compradorDAO`. Se não houver compradores cadastrados, exibe uma mensagem informando que ainda não existem compradores cadastrados.

15. `buscaComprador(CompradorDAO compradorDAO)`: Método que busca um comprador pelo nome, utilizando um objeto `CompradorDAO`. Ele percorre a lista de compradores e verifica se o nome informado pelo usuário corresponde a algum comprador. Se encontrar correspondência, exibe as informações do comprador. Caso contrário, exibe mensagens indicando se não há compradores cadastrados ou se nenhum comprador foi encontrado.

16. `buscaFornecedor(FornecedorDAO fornecedorDAO)`: Método que busca um fornecedor pelo nome, utilizando um objeto `FornecedorDAO`. Ele percorre a lista de fornecedores e verifica se o nome informado pelo usuário corresponde a algum fornecedor. Se encontrar correspondência, exibe as informações do fornecedor. Caso contrário, exibe mensagens indicando se não há fornecedores cadastrados ou se nenhum fornecedor foi encontrado.

17. `buscaProduto(ProdutoDAO produtoDAO)`: Método que busca um produto pelo nome, utilizando um objeto `ProdutoDAO`. Ele percorre a lista de produtos e verifica se o nome informado pelo usuário corresponde a algum produto. Se encontrar correspondência, exibe as informações do produto. Caso contrário, exibe mensagens indicando se não há produtos cadastrados ou se nenhum produto foi encontrado.

18. `listaEstoques(EstoqueDAO estoqueDAO)`: Método que lista todos os estoques cadastrados, utilizando um objeto `EstoqueDAO`. Ele obtém a lista de estoques e exibe as informações de cada estoque, incluindo o ID do estoque, o ID do produto e a quantidade. Se não houver estoques cadastrados, exibe uma mensagem indicando isso.

19. `buscaEstoque(EstoqueDAO estoqueDAO)`: Método que busca um estoque pelo ID do produto, utilizando um objeto `EstoqueDAO`. Ele solicita ao usuário que informe o ID do produto e busca o estoque correspondente. Se encontrar o estoque, exibe suas informações. Caso contrário, exibe uma mensagem informando que o estoque não foi encontrado.

20. `atualizaProduto(ProdutoDAO produtoDAO, FornecedorDAO fornecedorDAO)`: Método que atualiza as informações de um produto, utilizando objetos `ProdutoDAO` e `FornecedorDAO`. Ele solicita ao usuário o ID do produto a ser atualizado, exibe as informações atuais do produto e permite a atualização do nome, preço e fornecedor. Para isso, busca o fornecedor pelo ID informado e, se encontrado, atualiza as informações do produto. Ao final, exibe uma mensagem informando se os dados foram atualizados com sucesso ou se o fornecedor não foi encontrado.

21. `atualizaFornecedor(FornecedorDAO fornecedorDAO)`: Método que atualiza as informações de um fornecedor, utilizando um objeto `FornecedorDAO`. Ele solicita ao usuário o ID do fornecedor a ser atualizado e, se encontrado, permite a atualização do nome e CNPJ. Verifica se o CNPJ informado é válido e, caso seja, atualiza as informações do fornecedor. Ao final, exibe uma mensagem informando se os dados foram atualizados com sucesso.

22. `atualizaEstoque(EstoqueDAO estoqueDAO)`: Método que atualiza a quantidade de um estoque, utilizando um objeto `EstoqueDAO`. Ele solicita ao usuário o ID do produto

do estoque a ser atualizado e, se encontrado, permite a atualização da quantidade. Ao final, exibe uma mensagem informando se os dados

22. `atualizaEstoque(EstoqueDAO estoqueDAO)`: Método que atualiza a quantidade de um estoque, utilizando um objeto `EstoqueDAO`. Ele solicita ao usuário o ID do produto do estoque a ser atualizado e, se encontrado, permite a atualização da quantidade. Ao final, exibe uma mensagem informando se os dados foram atualizados com sucesso.

23. `removeComprador(CompradorDAO compradorDAO)`: Método que remove um comprador do sistema, utilizando um objeto `CompradorDAO`. Ele solicita ao usuário o ID do comprador a ser removido e, se encontrado, realiza a exclusão do comprador. Ao final, exibe uma mensagem informando se o comprador foi removido com sucesso.

24. `removeFornecedor(FornecedorDAO fornecedorDAO)`: Método que remove um fornecedor do sistema, utilizando um objeto `FornecedorDAO`. Ele solicita ao usuário o ID do fornecedor a ser removido e, se encontrado, realiza a exclusão do fornecedor. Ao final, exibe uma mensagem informando se o fornecedor foi removido com sucesso.

25. `removeProduto(ProdutoDAO produtoDAO)`: Método que remove um produto do sistema, utilizando um objeto `ProdutoDAO`. Ele solicita ao usuário o ID do produto a ser removido e, se encontrado, realiza a exclusão do produto. Ao final, exibe uma mensagem informando se o produto foi removido com sucesso.

27. O método venda realiza uma venda de produtos, utilizando instâncias de CompraDAO, ProdutoDAO e CompradorDAO. Ele permite adicionar produtos à venda, verificando se o comprador e o produto existem, e se a quantidade desejada está disponível no estoque. Os dados da venda são registrados e salvos no banco de dados.

28.O método listaVendas lista todas as vendas registradas no sistema, exibindo informações como o ID da venda, o preço total e o comprador.

29.O método atualizaComprador atualiza os dados de um comprador existente. Ele solicita o nome do comprador a ser atualizado, busca na lista de compradores e permite alterar o nome, CPF ou CNPJ, dependendo do tipo de comprador.

30.O método deletaComprador exclui um comprador existente, juntamente com todas as suas compras associadas. Ele busca o comprador na lista de compradores, remove-o do banco de dados e exclui as compras relacionadas.

Na main, existel com um menu interativo para gerenciar diferentes aspectos do sistema. O menu inclui opções para criar usuários, gerenciar produtos, estoques, fornecedores, vendas, compradores e usuários. Cada opção selecionada no menu chama um método correspondente para executar as operações relacionadas.

As principais funcionalidades do código são:

1. Gerenciamento de Produtos: Permite criar, listar, buscar, atualizar e excluir produtos.
2. Gerenciamento de Estoques: Permite listar, buscar e atualizar informações sobre os estoques.
3. Gerenciamento de Fornecedores: Permite criar, listar, buscar, atualizar e excluir fornecedores.
4. Gerenciamento de Vendas: Permite realizar vendas, listar vendas realizadas.

5. Gerenciamento de Compradores: Permite criar, listar, buscar, atualizar e excluir compradores.

6. Gerenciamento de Usuários: Permite criar, listar, buscar, atualizar e excluir usuários do sistema.

O programa utiliza um loop para exibir o menu repetidamente até que o usuário selecione a opção de encerrar (0). Cada opção selecionada no menu direciona o programa para um método correspondente, que realiza as operações relacionadas à opção selecionada.

Em resumo, o projeto de estoque apresenta um sistema robusto e modular que permite o gerenciamento eficiente do estoque de uma empresa. Com funcionalidades abrangentes, usabilidade amigável e foco na tomada de decisões, o sistema contribui para a otimização dos processos internos, o controle de custos e a satisfação dos clientes.