Pró-Reitoria Acadêmica
Escola de Educação, Tecnologia e Comunicação
Curso de Bacharelado em Ciência da Computação

Material de Construção

Autor: Ana Beatriz Cavalcante Amorim, Bárbara Eloi Dos Santos, Déborah Araujo Mendes, Erick Alves De Queiroz Martins, João Pedro Oliveira Alencar .

Orientador: Adam Smith Gontijo Brito de Assis.

Brasília - DF
2024

Material de Construção

Documento apresentado ao Curso de graduação Bacharelado em Ciência da Computação, da Universidade Católica de Brasília, como requisito parcial para obtenção da aprovação na disciplina de Engenharia de software.

Orientador: Prof. Adam Smith Gontijo Brito de Assis.

Brasília
2024

RESUMO

Referência: AMORIM, Ana Beatriz Cavalcante Amorim; DOS SANTOS, Bárbara Eloi dos Santos, MENDES, Déborah Araujo Mendes, MARTINS, Erick Alves de Queiroz Martins, ALENCAR, João Pedro Oliveira Alencar; Material de Construção , 2024. nr p. Bacharelado em Ciência da Computação,– UCB – Universidade Católica de Brasília, Taguatinga – DF, 2024.

Este trabalho tem como objetivo principal resolver um problema hipotético de uma loja de materiais de construção. O desafio consistia em substituir uma ficha de controle manual por uma solução automatizada, utilizando conhecimentos em banco de dados e programação orientada a objetos, visando otimizar o tempo e melhorar a eficiência dos processos.

Palavras-chave: ficha de controle, otimização, automatizar.

ABSTRACT

This work aims to address a hypothetical problem faced by a construction materials store. The challenge involved replacing a manual control sheet with an automated solution, leveraging knowledge of databases and object-oriented programming to optimize time and improve process efficiency.

Keywords: control sheet, optimization, automate.

SUMÁRIO

1 INTRODUÇÃO

O controle manual de informações ainda é uma prática comum em muitos pequenos negócios, como lojas de materiais de construção, onde a gestão de produtos e clientes frequentemente é realizada por meio de fichas preenchidas à mão. Este método, embora tradicional, apresenta diversas limitações, como risco de erros, demora nos processos e dificuldade para organizar e acessar os dados.

Diante desse cenário, este trabalho busca propor uma solução informatizada para otimizar o gerenciamento de uma loja de materiais de construção. A partir de um estudo sobre o funcionamento do sistema manual, que utilizava fichas de controle coladas em cadernos para registrar a retirada de produtos, foi desenvolvido um sistema automatizado utilizando conceitos de banco de dados e programação orientada a objetos.

O objetivo principal é substituir o sistema manual por uma ferramenta digital que permita registrar, organizar e gerenciar as informações de forma eficiente e segura. A solução visa não apenas reduzir o tempo necessário para as operações, mas também aumentar a confiabilidade e a precisão dos dados, oferecendo à loja um meio moderno e eficaz de gerenciar seus processos internos.

3 EXPLICAÇÃO DETALHADA

3.1 VALIDACAOVIEW

A classe **ValidacaoView**, representa a interface gráfica principal do sistema de validação de entrada de um material de construção. Tendo botões para em sua janela com essas três funcionalidades. Visualizar uma ficha após a validação da senha, inserir uma nova ficha no sistema e encerrar o programa, sendo a classe herdada do **JFrame.**

As variáveis usadas no código para compor a interface gráfica da classe **ValidacaoView.** Entre elas, o título da janela, representado pela variável Titulo, que é um componente do tipo **JLabel**. Esse título exibe o texto **"Validação de Entrada"** na parte superior da interface e é estilizado com fonte grande e em negrito para proporcionar maior destaque.

Além disso, dois painéis do tipo **JPanel** organizam os elementos da interface. O **PainelCentral** é responsável por dispor os botões **"Visualizar Ficha"** e **"Inserir Ficha"** em um layout de grade com espaçamento definido, enquanto o **PainelInferior** organiza o botão **"Sair"** em um layout centralizado. Assim como o título, essas variáveis também não são imutáveis.

Quanto aos botões, o código possui três principais do tipo **JButton**:

- **BotaoVisualizar** : após a validação da senha o usuário acessar a funcionalidade de visualização das fichas.

- **BotaoInserir** : o usuário acessa a funcionalidade de inserir uma nova ficha no sistema.

- **BotaoSair** : encerra o programa ao ser pressionado.

Os métodos usados no código para controlar e configurar o funcionamento da interface gráfica, tem como o construtor **ValidacaoView()** responsável pela configuração dos elementos gráficos como painéis e o layout , juntamente com os botões e suas ações a serem executadas. Esse método se trata de um construtor, então não recebe parâmetros e nem retorna valor.

Os botões da interface possuem ações específicas, como exemplo, o botão **"Visualizar Ficha"** exibe um campo de entrada para a validação da senha do administrador. Caso a senha seja **validada** com sucesso, a janela atual é fechada, e a funcionalidade de busca de fichas **(classe BuscaId)** é aberta. Caso contrário, uma mensagem de erro é exibida ao usuário. O botão "Inserir Ficha", por sua vez, fecha a janela atual e redireciona o usuário para a funcionalidade de inserção de fichas **(classe ClienteView).** Já o botão **"Sair"** simplesmente encerra a execução do programa, sendo assim, nenhum dos botões possui parâmetros ou retorna valores.

Atrelado a isso, o método principal **main (String[ ] args)** é utilizado para inicializar a aplicação gráfica, criando uma nova instancia da **classe ValidacaoView**. Ele recebe como parâmetro um **array de strings (args)**, não retornando nenhum valor esse método.

3.2 CLIENTEVIEW

A classe **ClienteView** tem como objetivo criar a interface gráfica de cadastro de clientes, que contém um construtor chamado **ClienteView()**, que não realiza nenhuma ação específica e serve apenas para inicializar a classe. O ponto de entrada da aplicação é o método **main(String[] args)**, que configura toda a interface gráfica. Esse método cria uma janela principal do tipo **JFrame** com o título **"Cadastro Cliente"** e utiliza o layout **BorderLayout** para organizar os componentes na tela. Dentro da janela, um **JLabel** exibe o título da tela, enquanto um painel do tipo **JPanel** organiza os campos de texto em uma grade **GridLayout** para a inserção dos dados do cliente, como ID, nome, CPF, RG, endereço, cidade e UF.

Os campos de entrada de dados são representados por **JTextField**, onde o usuário digita as informações necessárias, sendo cada campo de texto corresponde a um tipo específico de dado do cliente, como ID do cliente, nome, CPF, RG, entre outros. Além disso, há um painel adicional, **PainelBotao**, que contém um **JButton** chamado **"Enviar"**. Quando esse botão é clicado, uma ação é executada para fechar a janela de cadastro de cliente e abrir uma nova janela para o cadastro de produto.

A ação do botão **"Enviar"** é definida por um método que usa um **ActionListener** para capturar o clique no botão. Quando o botão é clicado, o método executa três ações: ele fecha a janela atual (Tela.dispose()), abre a janela de cadastro de produto criando uma instância de CadastroProdutoView e a torna visível, e exibe uma mensagem de sucesso ao usuário usando **JOptionPane.showMessageDialog**, informando que o login foi bem-sucedido

Além disso, as fontes e bordas aplicadas aos componentes da interface, como **JLabel** e **JButton**, são constantes e não mudam durante a execução do programa, garantindo um melhor estilo da interface.

3.3 CADASTROPRODUTOVIEW

A classe **CadastroProdutoView** herda de **JFrame**, a classe base para a criação de interfaces gráficas em Java. Através dessa herança, a classe consegue exibir uma janela contendo todos os componentes da interface, sendo responsável pela criação de todos os elementos gráficos que o usuário interage, configurando o layout, os campos de entrada de dados e o botão de envio.

O construtor da classe **CadastroProdutoView** é o responsável por inicializar a janela e todos os seus componentes. Dentro desse método, são realizadas as seguintes ações:

Configuração da janela o título é definido como **"Material de Construção",** e o tamanho da janela é ajustado para 600x600 pixels. O layout da janela é configurado com o uso do **BorderLayout**, que organiza os componentes em áreas distintas (norte, sul, centro, etc.). A criação do título utilizou um **JLabel** é criado para exibir o título **"Cadastro de Produto",** com uma fonte personalizada (Verdana, negrito, tamanho 22). Esse título é adicionado à parte superior da janela (área "North"). E a criação dos campos de entrada, o painel principal **JPanel Principal** utiliza um layout de grid (com 12 linhas e 2 colunas) para organizar os

campos de texto **JTextField** e os rótulos **JLabel** correspondentes. Cada rótulo descreve um campo (ID do Produto, Descrição, Quantidade, Preço Unitário, Preço Total), e ao lado de cada rótulo há um campo de texto onde o usuário pode inserir os dados. Esses campos são configurados com um tamanho de 20 colunas e uma fonte personalizada (Verdana, negrito, tamanho 16). Atrelado a isso, o botão "Enviar" é adicionado em um painel **JPanel PainelBotao**, e ao ser pressionado, ele executa uma ação definida por um evento anônimo. Este evento fecha a janela atual com o **método dispose()** e cria uma nova instância de **CompraView**, exibindo-a em seguida. Além disso, uma mensagem de sucesso é exibida usando **JOptionPane.showMessageDialog().**

3.4 COMPRAVIEW

O construtor da classe, **CompraView()**, é utilizado para configurar a interface gráfica inicial da aplicação. Primeiramente, ele define o título da janela como **"Material de Construção"** e estabelece a operação de fechamento da janela, o tamanho da janela é fixado em 600x600 pixels, e o layout é configurado como **BorderLayout** para organizar os componentes na tela. A janela exibe, na parte superior, um título **JLabel** com o texto **"Detalhes Compra",** utilizando a fonte Verdana em negrito e tamanho 22.

Dentro da janela, a classe cria um painel principal **JPanel** com o layout **GridBagLayout,** que organiza os elementos em uma grade. São adicionados três rótulos **JLabel** e campos de entrada de texto **JTextField** para que o usuário insira o ID da compra, o preço total e a data da compra. Cada rótulo é configurado com a fonte Verdana, tamanho 16 e negrito, e os campos de texto possuem 20 colunas. Esses campos recebem, respectivamente, os valores para o ID, o preço e a data da compra.

Na parte inferior da janela, há um painel adicional **JPanel** que contém um botão **JButton** chamado **"Enviar".** Este botão é configurado com a fonte Arial, tamanho 16, e tem um estilo de borda e foco personalizado. O botão possui um ouvinte de ação que, ao ser clicado, fecha a janela atual e abre uma nova janela de validação, representada pela classe **ValidacaoView**. Ao abrir a nova janela, uma mensagem de sucesso é exibida ao usuário, indicando que o login foi bem-sucedido.

3.5 BUSCARID

A classe **BuscaId** foi criada para desenvolver a interface com o objetivo de permitir que o usuário insira um ID e, ao clicar em um botão, o programa processe essa informação. Dentro dessa classe, há várias variáveis que definem os componentes gráficos da interface. A variável Titulo é um objeto do tipo **JLabel**, responsável por exibir o título da janela**, "Busca por ID".** Ela é pública, ou seja, pode ser acessada fora da classe e não é constante, permitindo alterações. Já a variável **PainelCentral** é um objeto do tipo JPanel, que organiza os componentes centrais da interface, como o campo de texto e o botão. Assim como o Titulo, é pública e pode ser modificada, a variável **LabelID** é outra instância de **JLabel**, que exibe o texto **"Insira o ID:"**, orientando o usuário sobre o que deve ser feito. Ela também é pública e alterável, a variável **CampoID**, que é do tipo **JTextField**, permite que o usuário insira o ID desejado, sendo igualmente pública e passível de alterações. A última variável, **BotaoEnviar**,

é um **JButton**, que aciona a busca pelo ID quando o usuário clica nele. Ela também é pública e pode ser modificada conforme necessário.

O construtor da classe, chamado **BuscaId(),** tem a responsabilidade de inicializar todos os componentes gráficos da interface. O layout da janela é definido no construtor, onde o Titulo é posicionado na parte superior da janela e o painel central é preenchido com o rótulo solicitando a entrada do ID, o campo de texto para o ID e o botão para enviar a informação.

Quanto aos métodos, **o CampoID.getText()** é utilizado para obter o texto inserido pelo usuário no campo de texto. Esse método não recebe parâmetros e retorna o texto, que é processado pelo **.trim()** para remover quaisquer espaços extras. O valor obtido é então armazenado na variável var2. O método **BotaoEnviar.addActionListener(...)** adiciona um ouvinte de evento ao botão "Entrar", de forma que, quando o botão for pressionado, a ação de capturar o texto do campo de texto será executada.

3.6 CLIENTEMT

A classe **ClienteMT** é responsável por armazenar as informações de um cliente, como seu identificador (ID), nome, RG, CPF, endereço, cidade e estado (UF). Ela contém variáveis de instância protegidas, ou seja, podem ser acessadas diretamente nas subclasses ou dentro do pacote. As variáveis são:

- **id_cliente**: um inteiro que representa o identificador único do cliente.

- **nome_cliente**: uma string que armazena o nome completo do cliente.

- **rg_cliente**: uma string que guarda o RG (registro geral) do cliente.

- **cpf_cliente**: uma string que contém o CPF (Cadastro de Pessoa Física) do cliente.

- **endereco_cliente**: uma string que armazena o endereço completo do cliente.

- **cidade_cliente**: uma string que contém a cidade onde o cliente reside.

- **uf_cliente**: uma string que representa a unidade federativa (estado) do cliente.

A classe possui métodos **get** e **set** para cada uma dessas variáveis. Os métodos **get** são usados para retornar o valor de uma variável, permitindo que se acesse as informações armazenadas. Já os métodos **set** são usados para definir ou alterar o valor das variáveis, recebendo o valor que será atribuído à variável correspondente.

O **construtor** padrão da classe **ClienteMT** não recebe parâmetros e cria um objeto com valores padrão. Para alterar ou acessar as informações do cliente, a classe disponibiliza os métodos **set** e **get** para cada variável. Por exemplo, para alterar o nome de um cliente, utiliza-se o método **setNome_cliente(String nome_cliente)**, passando o novo nome como parâmetro. Para acessar o nome, utiliza-se o método **getNome_cliente()**, que retorna o valor armazenado na variável **nome_cliente.**

3.7 COMPRASMT

A classe **CompraMT** tem como objetivo representar o processo de compra de produtos, associando um **cliente** e um **produto** a uma compra, além de armazenar atributos como o preço total da compra e a data em que foi realizada. Essa classe possui alguns atributos importantes: um objeto **ClienteMT** para representar o cliente associado à compra, um objeto **ProdutoMT** para representar o **produto**, e três variáveis do tipo primitivo: **id_compra** (que é um identificador único para a compra), **preco_total_compra** (que armazena o valor total da compra) e **data_compra** (que armazena a data em que a compra foi realizada).

A classe possui um **construtor** padrão **public CompraMT()**, que é usado para criar um novo objeto dessa classe sem a necessidade de passar parâmetros. Além disso, a classe oferece **métodos getters e setters** para acessar e modificar os atributos. Esses métodos são: **getId_compra() e setId_compra(int id_compra),** que permitem acessar e modificar o identificador da compra; **getPreco_total_compra()** e **setPreco_total_compra(double preco_total_compra)**, que manipulam o **preço total** da compra; e **getData_compra()** e **setData_compra(Date data_compra)**, que controlam a data da compra.

O método mais complexo da classe é o **public void CadastrarCompra(ClienteMT cliente, ProdutoMT produto)**, que tem como principal função cadastrar uma nova compra no banco de dados. Ele recebe dois parâmetros: um objeto cliente da classe **ClienteMT** e um objeto produto da classe **ProdutoMT**. O processo de cadastro envolve várias etapas: primeiro, o método cria uma **conexão** com o banco de dados utilizando a classe **ConnectionFactory**. Em seguida, ele insere os dados do cliente na tabela cliente e obtém o ID gerado para esse cliente. Depois, insere os dados do produto na tabela produto e também obtém o ID gerado para o produto. Por fim, insere os dados da compra na tabela compra, associando o cliente e o produto registrados anteriormente. Se todas as operações forem realizadas com sucesso, a compra é cadastrada e uma mensagem de confirmação é exibida para o usuário. Caso contrário, o método realiza o rollback da transação e exibe uma mensagem de erro.

Dessa forma, a classe **CompraMT** tem um papel crucial no sistema, sendo responsável por gerenciar e registrar as compras no banco de dados, associando clientes e produtos de maneira eficiente.

3.8 PRODUTOMT

A classe ProdutoMT é responsável por representar um produto dentro do sistema, com atributos e métodos que permitem o armazenamento e manipulação das informações relacionadas ao produto, como o identificador, a descrição, a quantidade disponível, o preço unitário e o preço total.

- id_produto: Um atributo do tipo int que armazena o identificador único de cada produto. Esse valor é utilizado para distinguir um produto de outro.

- descricao_produto: Um atributo do tipo String que contém a descrição textual do produto, como seu nome ou outras informações que o caracterizam.

- quantidade_produto: Um atributo do tipo int que representa a quantidade disponível em estoque de um determinado produto.

- preco_unitario_produto: Um atributo do tipo double que armazena o valor de um único item do produto. Ele é utilizado para calcular o preço total do produto com base na quantidade adquirida.

- preco_total_produto: Um atributo do tipo double que representa o valor total do produto, calculado multiplicando a quantidade pelo preço unitário.

O **construtor padrão** da classe, ProdutoMT(), não recebe parâmetros e apenas inicializa o objeto, permitindo que os valores dos atributos sejam configurados posteriormente.

A classe possui métodos para acessar e modificar os valores dos atributos. Esses métodos são conhecidos como **getters** (métodos de acesso) e **setters** (métodos de modificação). Para cada atributo, há um método de acesso (getter) que retorna o valor do atributo, e um método de modificação (setter) que define o valor do atributo. Abaixo estão os métodos da classe:

- getId_produto(): Retorna o valor do atributo id_produto, que é o identificador do produto.

- setId_produto(int id_produto): Define o valor do atributo id_produto com o valor recebido como parâmetro.

- getDescricao_produto(): Retorna o valor do atributo descricao_produto, que é a descrição textual do produto.

- setDescricao_produto(String descricao_produto): Define o valor do atributo descricao_produto com o valor recebido como parâmetro.

- getQuantidade_produto(): Retorna o valor do atributo quantidade_produto, que é a quantidade disponível do produto.

- setQuantidade_produto(int quantidade_produto): Define o valor do atributo quantidade_produto com o valor recebido como parâmetro.

- getPreco_unitario_produto(): Retorna o valor do atributo preco_unitario_produto, que é o preço unitário de um produto.

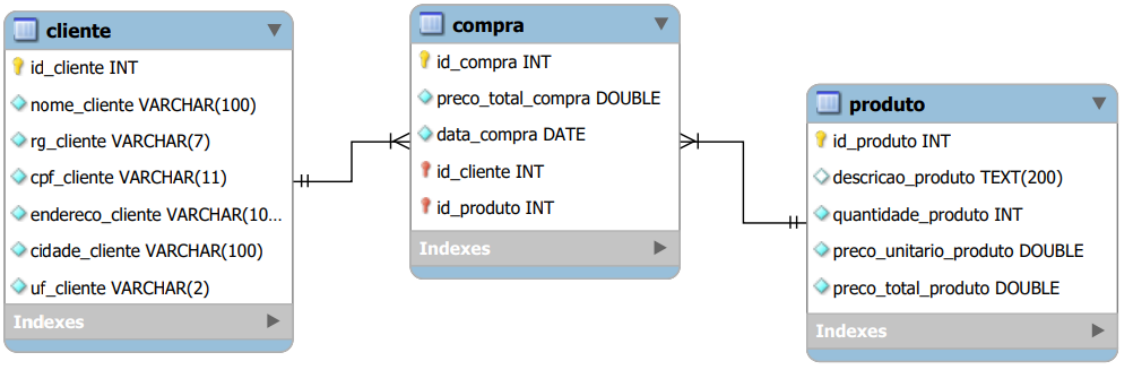
- setPreco_unitario_produto(double preco_unitario_produto): Define o valor do atributo preco_unitario_produto com o valor recebido como parâmetro.

- getPreco_total_produto(): Retorna o valor do atributo preco_total_produto, que é o preço total do produto, calculado com base na quantidade e no preço unitário.

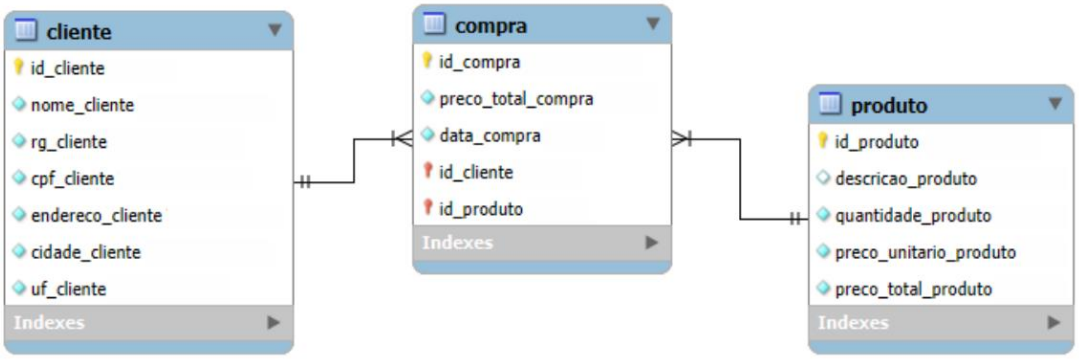
- setPreco_total_produto(double preco_total_produto): Define o valor do atributo preco_total_produto com o valor recebido como parâmetro.

4 MODELAGEM DO BANCO

4.1 MODELAGEM FÍSICA



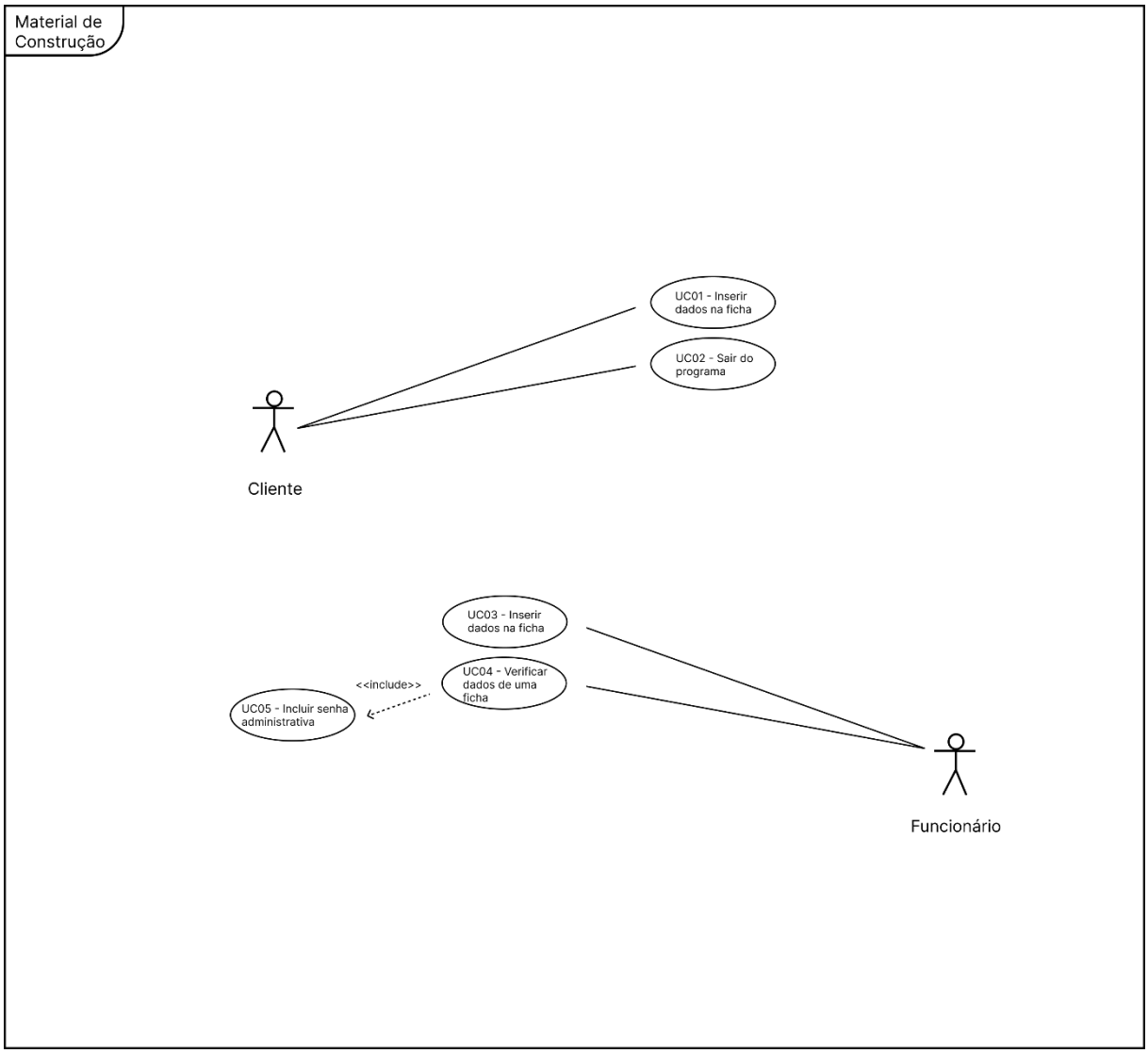
Z4.2 MODELAGEM LÓGICA



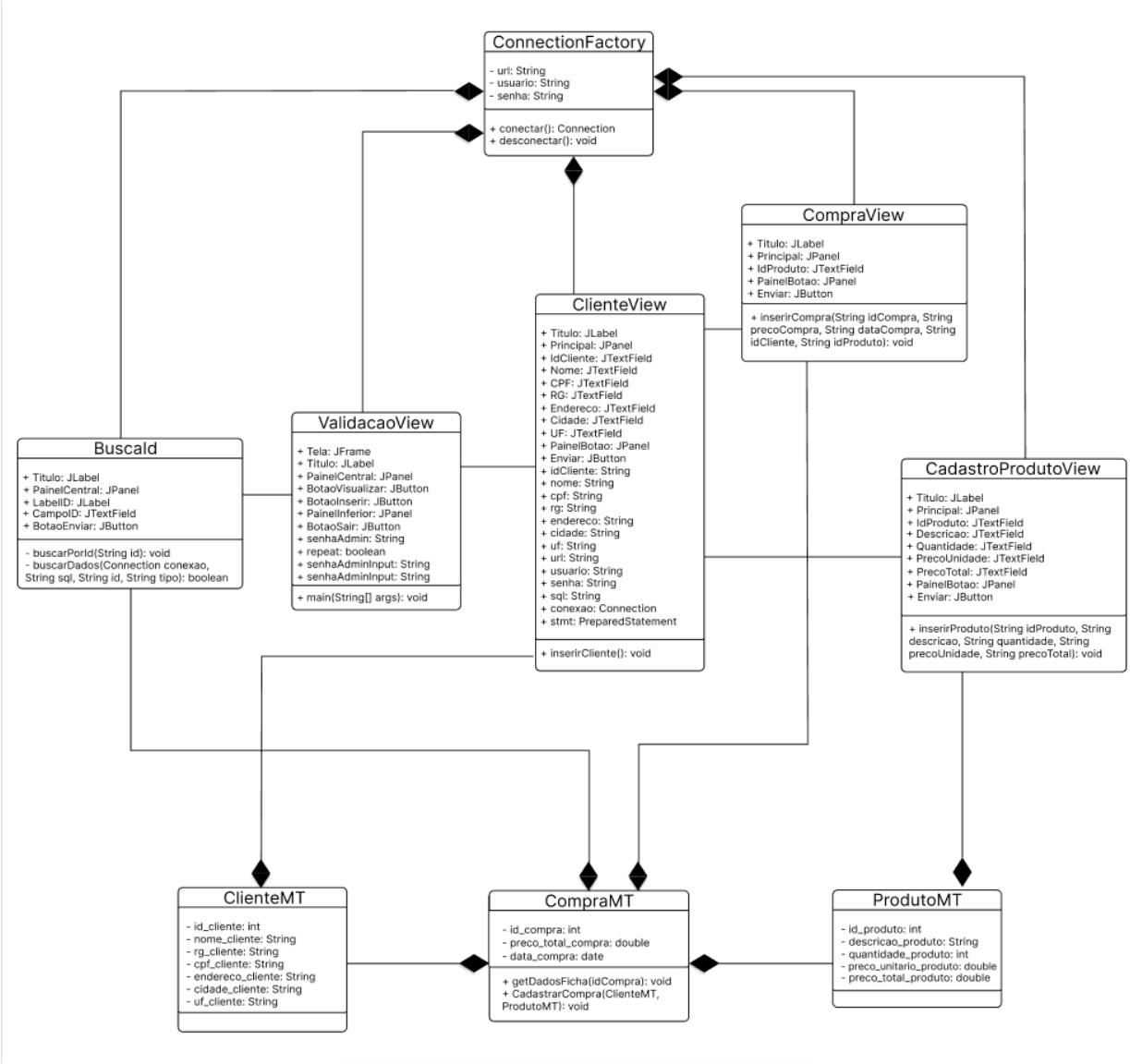
4.3 MODELAGEM CONCEITUAL

5 DIAGRAMAS

5.1 DIAGRAMA DE CASO DE USO



5.2 DIAGRAMA DE CLASSE



6 CONCLUSÃO

A implementação de uma solução informatizada para substituir o controle manual de informações em uma loja de materiais de construção demonstrou ser uma abordagem eficiente e eficaz. Com o uso de banco de dados e programação orientada a objetos, foi possível desenvolver um sistema que otimiza o registro, organização e gerenciamento de dados, atendendo às necessidades específicas da loja.

O sistema proposto trouxe melhorias significativas, como a redução do tempo necessário para realizar operações, maior precisão nos registros e maior segurança na gestão das informações. Além disso, a automatização dos processos contribuiu para a modernização do negócio, aumentando a confiabilidade das transações e a satisfação dos clientes.