

**UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS PATO BRANCO
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE
SISTEMAS**

ANDRÉ LUIS COLLA

**DESENVOLVIMENTO DE SISTEMA DE CONTROLE DE ESTOQUE E
PRODUÇÃO PARA MADEIREIRAS**

**PATO BRANCO
2013**

ANDRÉ LUIS COLLA

**DESENVOLVIMENTO DE SISTEMA DE CONTROLE DE ESTOQUE E
PRODUÇÃO PARA MADEIREIRAS**

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. Robison Cris Brito

**PATO BRANCO
2013**

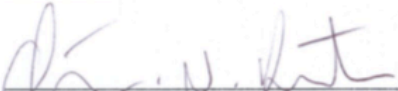
ATA Nº: 214

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DO ALUNO ANDRÉ LUIS COLLA.

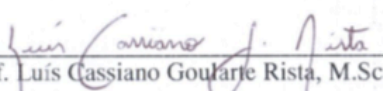
Às 15:26 hrs do dia 18 de setembro de 2013, Bloco V da UTFPR, Câmpus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Robison Cris Brito (Orientador), Géri Natalino Dutra (Convidado) e Luís Cassiano Goularte Rista (Convidado), para avaliar o Trabalho de Diplomação do aluno André Luis Colla, matrícula 995827, sob o título **Desenvolvimento de Sistema de Controle de Produção e Estoque para Madeireiras**; como requisito final para a conclusão da disciplina Trabalho de Diplomação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, COADS. Após a apresentação o candidato foi entrevistado pela banca examinadora, e a palavra foi aberta ao público. Em seguida, a banca reuniu-se para deliberar considerando o trabalho **APROVADO**. Às 15:54 hrs foi encerrada a sessão.



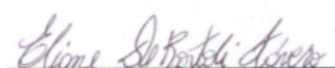
Prof. Robison Cris Brito, M.Sc.
Orientador



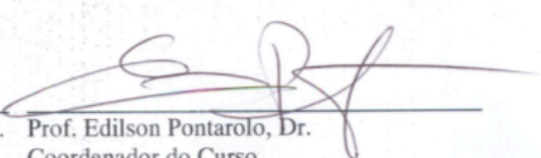
Prof. Géri Natalino Dutra, M.Sc.
Convidado



Prof. Luís Cassiano Goularte Rista, M.Sc.
Convidado



Prof. Eliane Maria De Bortoli Fávero, M.Sc.
Coordenadora do Trabalho de Diplomação



Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

AGRADECIMENTOS

Primeiramente a Deus, pois sem Ele não poderia chegar aqui, no qual deposito toda minha confiança.

À minha mãe, que fez de tudo o que estava ao seu alcance para me auxiliar na caminhada, com apoio, amor, compreensão, carinho e companheirismo, principalmente na ausência do meu pai.

Meu querido pai, o qual infelizmente não está mais aqui entre nós. Por ele e por minha mãe que pretendo terminar este curso.

À minha namorada, a qual me dá forças para seguir em frente e superar os desafios.

A estas pessoas maravilhosas agradeço todos os dias pelo apoio e inspiração que sempre levarei comigo para toda a vida.

Aos professores Géri Natalino Dutra e Robison Cris Brito, pela colaboração, compreensão e dedicação para comigo.

A todos os professores do curso, pela paciência de repassar o conhecimento.

Em especial agradeço ao amigo Carlos Roberto Zucchi, sendo que este repassou dicas, informações e material para auxiliar no desenvolvimento deste trabalho.

Também à Instituição, pela oportunidade.

RESUMO

COLLA, André Luis. Desenvolvimento de sistema de controle de estoque e produção para madeiras. 2013. 65 f. Monografia de Trabalho de Diplomação – Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná. Câmpus Pato Branco. Pato Branco, 2013.

Este trabalho foi realizado com o intuito de elaborar uma solução computacional para o controle do processo de produção e estoque de uma empresa de madeiras, utilizando o estudo de modelagem de sistemas e analisando a ferramenta NeoDatis ODB para o controle dos dados e a linguagem de programação Java para o desenvolvimento do sistema. Inicialmente foi realizado o estudo para identificação das funcionalidades que o sistema deveria possuir, para que este atenda as necessidades do cliente, sendo então realizado o processo de Análise e Modelagem dos processos internos de uma empresa de produção e estoque de madeiras. Este processo inicial foi focado no estudo que foi realizado como requisito para a matéria de Estágio Supervisionado, onde foram levantados os requisitos principais do projeto, bem como seus relacionamentos e comportamentos, utilizando-se da Linguagem Unificada de Modelagem (UML) para a representação dos mesmos em forma de diagramas. Após foi realizada a codificação do sistema utilizando a metodologia de desenvolvimento de software orientada a objetos, a qual engloba os processos de Análise, Projeto e Implementação, sendo então utilizada a linguagem de desenvolvimento Java. Para o armazenamento dos dados foi optado por utilizar uma ferramenta alternativa, a qual possui utilização simples e permite utilizar a linguagem nativa de programação, no caso Java, para a persistência dos objetos da aplicação, sem necessidade de conversão de dados ou o uso de comandos SQL. Por fim, com o sistema desenvolvido, foi feita uma avaliação do mesmo, sendo possível verificar que os requisitos levantados foram atendidos de forma satisfatória, e este está pronto para ser implementado no cliente.

Palavras-chave: Controle de Estoque. Java Desktop. Persistência Orientada a Objeto. Análise Orientada a Objeto.

ABSTRACT

COLLA, André Luis. Development of an inventory control and production system for logging. 2013. 65 f. Monografia de Trabalho de Diplomação – Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Campus Pato Branco. Pato Branco, 2013.

This work was carried out in order to develop a computational solution for process control of production and stock of a company woods, using the study modeling and analyzing tool NeoDatis ODB for the control of the data and the Java programming language for system development. Initially the study was conducted to identify the features that the system should have, so that it meets the customer's needs, and then carried out the process of analysis and modeling of the internal processes of a company's production and inventory woods. This initial process was focused on the study that was conducted as a requirement for the issue of Supervised, where they were raised the main requirements of the project, well as their relationships and behaviors, using de Unified Modeling Language (UML) for the representation thereof in the form of diagrams. Was held after the encoding system using the methodology of developing object-oriented software, which encompasses the processes of Analysis, Design and Implementation, and then used the Java development language. For storage of the data was chosen to use an alternative tool, which features simple and allows you to use the native programming language, case in Java, for the persistence of the application objects, without conversion or data using SQL commands. Finally, with the system developed, an evaluation was made of the same, is possible to verify that the requirements have been met raised satisfactorily, and this is ready to be implemented on the client.

Keywords: Inventory Control. Java Desktop. Persistence Object-Oriented. Object-Oriented Analysis.

LISTA DE FIGURAS

Figura 1 - Objetos de duas classes.....	21
Figura 2 - Duas classes: cães e gatos.....	22
Figura 3 - Hierarquia de Classes.....	22
Figura 4 - Tela de inicialização do Visual Paradigm	28
Figura 5 - Tela de inicialização do NetBeans IDE	30
Figura 6 - Tela de configuração de conexão com BD do NeoDatis.....	32
Figura 7 - Diagrama de Caso de Uso	36
Figura 8 - Diagrama de Caso de Uso (CRUD).....	37
Figura 9 - Diagrama de Classes (parte I).....	41
Figura 10 - Diagrama de Classes (parte II).....	42
Figura 11 - Diagrama de Classes (parte III).....	42
Figura 12 - Pacotes do Projeto	43
Figura 13 - Tela de Login do sistema	53
Figura 14 - Mensagem de validação do login.....	53
Figura 15 - Menu principal do sistema.....	54
Figura 16 - Tela de Listagem de Compras.....	55
Figura 17 - Tela de Cadastro de Compras.....	56
Figura 18 - Mensagem para seleção de compra da lista	57
Figura 19 - Mensagem de Nota Fechada	57
Figura 20 - Mensagem de seleção de registro para exclusão.....	59
Figura 21 - Mensagem de confirmação de exclusão	59
Figura 22 - Mensagem para selecionar uma compra	60
Figura 23 - Tela de Cadastro de Itens da Compra	60
Figura 24 - Mensagem de confirmação para fechar nota.....	61
Figura 25 - Tela de Listagem de Itens da Compra	62

LISTA DE QUADROS

Quadro 1 - Exemplo de Atribuição de valores para Atributos.....	20
Quadro 2 - Atributos de classe Cachorro (1).....	20
Quadro 3 - Atributos de classe Cachorro (2).....	21
Quadro 4 - Classe Conexao.java.....	47
Quadro 5 - Classe Fabrica.java.....	47
Quadro 6 - CriaBanco.java - Produção.....	48
Quadro 7 - Entidade Producao.java.....	50
Quadro 8 - Controladora ProducaoControle.java.....	52
Quadro 9 - Inserir novo registro.....	55
Quadro 10 - Código de alteração de registro.....	57
Quadro 11 - Código de exclusão de registro.....	58
Quadro 12 - Código para Inserir Item.....	60
Quadro 13 - Código para Visualizar registros cadastrados.....	61

LISTA DE ABREVIATURAS E SIGLAS

AOO	Análise Orientada a Objetos
API	<i>Application Programming Interface</i>
BDOO	Bando de Dados Orientado a Objetos
CRUD	<i>Create, Read, Update and Delete</i>
HTML	<i>Hyper Text Markup Language</i>
IDE	<i>Integrated Development Environment</i>
ODB	<i>Object Database</i>
OID	<i>Object Identifier</i>
OMT	<i>Object Modeling Technique</i>
OO	Orientado a Objetos
PHP	<i>Personal Home Page</i>
SGBD	Sistema Gerenciador de Banco de Dados
SQL	<i>Structured Markup Language</i>
UML	<i>Unified Modeling Language</i>
XML	<i>Extensible Markup Language</i>

SUMÁRIO

1 INTRODUÇÃO.....	11
1.1 CONSIDERAÇÕES INICIAIS	11
1.2 OBJETIVOS	13
1.2.1 Objetivo Geral.....	13
1.2.2 Objetivos Específicos.....	13
1.3 JUSTIFICATIVA	13
1.4 ESTRUTURA DO TRABALHO	14
2 REFERENCIAL TEÓRICO.....	16
2.1 ORIENTAÇÃO A OBJETOS.....	16
2.1.1 Análise Orientada a Objetos	17
2.1.2 Programação Orientada a Objetos	18
2.1.3 Benefícios da Orientação a Objetos.....	18
2.2 CONCEITOS DA ORIENTAÇÃO A OBJETOS	19
2.2.1 Objeto.....	19
2.2.2 Atributos e Métodos.....	19
2.2.3 Classes.....	20
2.2.4 Hierarquia de classes.....	22
2.2.5 Encapsulamento	23
2.2.6 Persistência	23
2.2.7 Herança	24
2.3 MODELO DE OBJETOS	25
2.4 BANCO DE DADOS ORIENTADO A OBJETOS	25
3 MATERIAIS E MÉTODO UTILIZADO.....	28
3.1 VISUAL PARADIGM 8.2.....	28
3.2 LINGUAGEM DE PROGRAMAÇÃO JAVA.....	29
3.3 IDE NETBEANS 7.1.1	30
3.4 BANCO DE DADOS NEODATIS ODB	31
3.5 METODOLOGIA ORIENTADA A OBJETOS.....	32
4 MODELAGEM DO SISTEMA.....	35
4.1 APRESENTAÇÃO DO SISTEMA	35
4.2 MODELAGEM DO SISTEMA.....	35
4.2.1 Diagrama de Casos de Uso	36
4.2.2 Casos de Uso.....	38
4.2.3 Diagrama de Classes	41
5 DESENVOLVIMENTO DO SISTEMA.....	43
5.1 ORGANIZAÇÃO DAS CLASSES DO PROJETO	43
5.2 PADRÃO DE NOMENCLATURAS DAS VARIÁVEIS	45
5.3 CRIAÇÃO DO BANCO DE DADOS.....	45
5.3.1 Classe Conexao.java	46
5.3.2 Classe Fabrica.java	47
5.3.3 Classe CriaBanco.java	48
5.3.4 Entidade Producao.java.....	49
5.3.5 Controladora ProducaoControle.java.....	51
5.3.6 Fronteira Login.java.....	52
5.3.7 Fronteira MenuPrincipal.java	53
5.3.8 Fronteira Compra.java	54

6 CONCLUSÃO	10
REFERÊNCIAS	63
	64

1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais com uma visão geral do trabalho, os objetivos, a justificativa e a organização do texto.

1.1 CONSIDERAÇÕES INICIAIS

Com a evolução das tecnologias de informação, os processos que envolvem controle e manipulação de dados começaram a ser implementados de maneira que fosse possível controlar um volume cada vez maior destas informações. Para que isso seja possível, foram desenvolvidos mecanismos que possibilitam que estes dados sejam manipulados por sistemas informatizados, os quais são chamados *softwares*.

Os dados de produção e controle de estoque apresentados de maneira prática e precisa auxiliam na identificação das reais necessidades de uma empresa para atingir seus objetivos e atender os clientes.

Atualmente existem diversas ferramentas para o desenvolvimento destas soluções computacionais, sendo que as empresas estão cada vez mais aderindo ao uso das tecnologias para otimizar os processos internos de controle, porém com a necessidade de se ter uma solução imediata, muitas vezes os desenvolvedores destes sistemas acabam não realizando as etapas iniciais do processo de desenvolvimento: a Análise e Modelagem, etapas consideradas fundamentais para que o sistema atenda as reais necessidades do cliente.

Ao desenvolver um *software* sem o uso das etapas iniciais, é comum que o cliente ao se deparar com o uso rotineiro do sistema acabe percebendo que algumas rotinas não são exatamente a solução que este precisava, e neste ponto é comum que haja reclamações por parte do cliente, pois este teve o custo de adquirir um produto ao qual não supre as suas reais necessidades. Com isso, para que o

sistema possa realizar o processo que o cliente precisa, os desenvolvedores precisam ajustar o produto, o que pode acarretar em mais custos para o cliente, e demandar tempo para a correção do sistema, o que gera desconforto com a equipe de desenvolvimento e o cliente, o qual muitas vezes fica com uma imagem negativa do produto.

Para evitar que isso ocorra, é preciso que as etapas iniciais tenham a devida atenção. Assim, primeiramente é importante que a equipe de desenvolvimento conheça o processo interno da empresa, recolhendo informações dos funcionários para saber quais são as reais necessidades que o sistema precisa atender para que o produto final seja realmente o que o cliente espera, deixando de lado informações que não são tão relevantes para o sistema.

Neste trabalho foi empregada a metodologia de desenvolvimento com base na orientação a objetos, sendo que foram realizadas as etapas de análise e modelagem dos requisitos do sistema, identificando nos processos internos de controle de uma maneira quais funcionalidades o sistema precisa oferecer para empresas do ramo. Este estudo inicial foi aprofundado com a elaboração do trabalho apresentado como requisito para a matéria de Estágio Supervisionado, sendo que estes dados foram utilizados neste projeto para a realização do desenvolvimento do sistema.

Tendo como base a análise e modelagem realizadas, este trabalho foi elaborado visando utilizar esses conhecimentos para a implementação de um *software* que atenda aos requisitos identificados. Para tanto, foram utilizadas as tecnologias de desenvolvimento orientado a objetos, tanto para a linguagem de programação quanto para o banco de dados, visando avaliar o comportamento de um sistema desenvolvido com estes recursos.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Realizar o desenvolvimento de um sistema para gerenciamento de produção e estoque de madeiras utilizando a metodologia de desenvolvimento de *software* orientada a objetos, fazendo o uso da ferramenta *NeoDatis ODB* para armazenamento de dados.

1.2.2 Objetivos Específicos

- Identificar os principais requisitos que o projeto deve atender para satisfazer a necessidade do cliente;
- Utilizar a ferramenta *NetBeans* e a linguagem Java para o desenvolvimento do *software*;
- Analisar o funcionamento do banco de dados *NeoDatis ODB*;
- Implementar telas de cadastro, controle de movimentações de estoque para o sistema a ser apresentado.

1.3 JUSTIFICATIVA

A escolha de realizar o desenvolvimento de um aplicativo que possibilite o controle de produção e estoque de madeiras ocorreu pela oportunidade de

desenvolver uma solução computacional para indústrias do ramo, possibilitando as mesmas uma maneira mais objetiva e fácil de controle dos processos diários destas empresas.

Para que o controle de dados da produção e estoque seja realizado de forma a auxiliar os processos internos de uma madeireira, existem várias etapas que podem ser automatizadas. Dentre estas se destacam: o processo de controle das toras utilizadas na laminação; o processo de classificação das lâminas fabricadas e das adquiridas de outras empresas; o controle das matérias-primas utilizadas para a produção das chapas de compensado; e a movimentação de entrada e saída dos produtos do estoque.

Tendo em vista da possibilidade de integrar os processos de uma madeireira a um sistema informatizado, foi pensado em realizar o desenvolvimento de um sistema que auxilie no controle dessas informações, o que irá proporcionar ao gerente de uma madeireira vários benefícios, o qual poderá realizar o controle de suas atividades diárias em menor tempo, possibilitando este a utilizar o tempo de maneira a dedicar-se no melhoramento dos processos internos da empresa, buscando novas e melhores técnicas de negócio e atividades gerenciais, visando também o contato com novos clientes e melhorando o atendimento aos mesmos, para uma maior satisfação de ambas as partes.

Sendo assim, o desenvolvimento deste trabalho pretende viabilizar a informatização do controle do processo de produção e estoque de madeireiras utilizando-se as tecnologias de desenvolvimento para *desktop* em Java e banco de dados com base na ferramenta NeoDatis ODB.

1.4 ESTRUTURA DO TRABALHO

Este relatório está organizado em capítulos, sendo que este é o primeiro, onde são apresentadas as ideias iniciais do projeto, com os objetivos e a justificativa.

No capítulo 2 será apresentado o referencial teórico do trabalho com base na orientação a objetos e no modelo de diagramas da UML. No capítulo 3 contém os materiais e o método utilizado para a realização do trabalho. O capítulo 4 contém a modelagem realizada do sistema. O capítulo 5 apresenta os métodos utilizados para o desenvolvimento do sistema. Por fim o capítulo 6 apresenta a conclusão do trabalho. Após são apresentadas as referências bibliográficas utilizadas como base para os textos.

2 REFERENCIAL TEÓRICO

Este capítulo contém o referencial teórico do trabalho. O foco é a utilização dos conceitos de análise e desenvolvimento Orientados a Objetos.

2.1 ORIENTAÇÃO A OBJETOS

Para o entendimento da modelagem orientada a objetos, é preciso compreender primeiramente o paradigma da orientação a objetos, seus princípios básicos, estruturas e relacionamentos.

Um objeto pode ser entendido como uma “coisa” física, por exemplo, um carro, uma casa ou uma moto. Pode-se definir objeto como uma equação ou uma conta bancária, neste caso puramente mental, pois não existiria uma “coisa” física que possa impressionar nossos sentidos para que estes o percebam como um objeto físico (CORREIA, 2006).

A proposta de orientação a objetos é representar da forma mais fiel possível as situações do mundo real nos sistemas computacionais. No caso, o mundo pode ser considerado como um todo, o qual é composto por vários objetos, e estes possuem interação entre si. Da mesma forma a orientação a objetos não considera os sistemas computacionais como uma coleção estruturada de processos, mas sim, como uma coleção de objetos que possuem interação uns com os outros (CORREIA, 2006).

Os objetos computacionais serão sempre simplificações dos objetos reais, e para que estes objetos possam interagir dentro de um sistema de computador de maneira a representar a interação dos objetos reais com o mundo real, estes precisam saber como agir perante uma ação. O comportamento de um objeto não é

determinado pelo sistema, e sim por ele próprio, o qual precisa ter a informação necessária para que possa saber como comportar-se diante de uma situação.

Pode-se concluir que, os objetos computacionais são estruturas de programação que contêm as informações e os comportamentos que representam um objeto dentro de um sistema (CORREIA, 2006).

2.1.1 Análise Orientada a Objetos

O processo de Análise Orientada a Objetos (AOO) consiste em identificar os objetos que fazem parte de um sistema e a maneira que se comportarão perante determinadas situações.

Esse processo de análise consiste em identificar quais objetos existem dentro do universo que se pretende automatizar e quais são os atributos e ações desses objetos (CORREIA, 2006). Desta maneira, é possível ao analista repassar o entendimento que ele possui da forma como o sistema funciona para o usuário leigo (CORREIA, 2006), possibilitando uma maior interação entre desenvolvedores e cliente.

O modelo de Análise Orientada a Objetos retrata objetos que representam um domínio de aplicações específico, juntamente com diversos relacionamentos estruturais e de comunicação, o qual serve para formalizar a “visão” do mundo real no qual o software será construído, estabelecendo os objetos que servirão como as principais estruturas organizacionais do sistema de software (YOURDON, 1999).

2.1.2 Programação Orientada a Objetos

Uma vez que os objetos possuem comportamentos em determinadas situações, para que os objetos computacionais possam realizar tais comportamentos corretamente, é preciso que sejam utilizadas estruturas de dados que os simulem, facilitando a programação pelo uso de uma metodologia unificada para análise e programação. Os objetos que foram identificados e modelados pelo analista podem ser implementados da mesma maneira que foram projetados, não sendo necessário traduzir as informações dos diagramas de análise para estrutura de dados e de programação, da forma que ocorre na análise estruturada (CORREIA, 2006).

Assim, a programação orientada a objetos consiste em utilizar objetos computacionais para implementar as funcionalidades de um sistema (CORREIA, 2006).

2.1.3 Benefícios da Orientação a Objetos

Na criação de um objeto por um desenvolvedor, este define quais serão as tarefas que o objeto irá desempenhar, tarefa esta que pode ser complexa, porém que os demais desenvolvedores não precisam saber como ela é efetuada, sendo preciso apenas acessar este objeto para realizar tal tarefa. Caso seja necessário mudar a forma com que o objeto realize a tarefa, os outros desenvolvedores não precisarão alterar seus programas para continuar acessando o novo comportamento, o que representa um benefício considerável de produtividade.

Uma das principais vantagens da orientação a objetos é reunir em uma mesma estrutura os dados e processos que são executados sobre esses objetos, o que permite melhor organização do programa (CORREIA, 2006).

2.2 CONCEITOS DA ORIENTAÇÃO A OBJETOS

A seguir serão apresentados alguns conceitos do processo de desenvolvimento OO (Orientação a Objetos), para melhor compreensão.

2.2.1 Objeto

Um objeto pode ser definido como representações do mundo real, ou seja, do mundo físico tal qual se tem conhecimento (CORREIA, 2006). Dessa forma, os objetos são utilizados no desenvolvimento de sistemas para que as situações do mundo real possam ser representadas em forma de dados computacionais, possibilitando que o comportamento de cada “coisa” seja controlado por um sistema informatizado.

Por exemplo, a representação de uma pessoa em um sistema pode ser considerada como um objeto. Esse objeto possui características próprias, as quais pertencem somente a ele, como nome, idade, tipo sanguíneo, cor dos cabelos, altura, peso, etc. Essas características são definidas como atributos (CORREIA, 2006).

2.2.2 Atributos e Métodos

A definição de atributos de objetos pode ser definida como as “variáveis” ou “campos” que guardam diferentes valores conforme as características que esses objetos podem possuir (CORREIA, 2006). A seguir segue um exemplo do objeto pessoa citado anteriormente:

Quadro 1 - Exemplo de Atribuição de valores para Atributos

Nome:	André Luis
Idade:	21
Altura:	1,74
Peso:	87
Cor do cabelo:	Castanho escuro
Cor dos olhos:	Castanho
Tipo sanguíneo:	O+

Seguindo a demonstração de atributos do objeto pessoa, estes mudarão de valor com processos externos e internos, sendo que para alterá-los será preciso que um evento seja disparado para que seja feita a mudança nas informações do objeto (CORREIA, 2006). Sendo assim, esses eventos que possibilitam realizar ações no objeto são definidas como métodos (CORREIA, 2006). Dessa forma, tudo o que o objeto faz é realizado por um método, e com essas ações o objeto pode interagir com os demais objetos.

Para o objeto pessoa citado, é possível definir alguns métodos, por exemplo caminhar, falar, correr, pular, dançar, entre outros.

2.2.3 Classes

Para a definição de classes, é possível utilizar cachorros como exemplo (CORREIA, 2006). Os quadros a seguir exemplificam os atributos da classe exemplificada:

Quadro 2 - Atributos de classe Cachorro (1)

Cachorro	
Nome:	Bradok

Idade:	2 anos
Comprimento dos pelos	Curto
Cor dos pelos	Marron e Branco
Cor dos olhos	Castanhos
Peso	6 kg

Quadro 3 - Atributos de classe Cachorro (2)

Cachorro	
Nome:	Rex
Idade:	4 anos
Comprimento dos pelos	Curto
Cor dos pelos	Preto, marrom e branco
Cor dos olhos	Castanhos
Peso	8 kg

Comparando esses dois cães, são identificados os mesmos atributos, pois se tratam de objetos da mesma classe, porém com valores diferentes.

Já na figura 1, podem-se distinguir duas categorias diferentes de animais: cães e gatos. Há vários objetos de duas classes diferentes. Assim, classe é um método e todos os seus objetos têm os mesmos atributos (mesmo com valores distintos) e métodos.



Figura 1 - Objetos de duas classes

Fonte das imagens:

Cão: <http://guiaveterinariopetropolis.blogspot.com.br/2013_01_01_archive.html>

Gato: <<http://mundolouco.net/gatos-podem-prever-sua-morte/>>

Na figura 2, cães e gatos são apresentados separadamente, cada um na classe a que pertence.



Figura 2 - Duas classes: cães e gatos

2.2.4 Hierarquia de classes

Analisando o exemplo citado anteriormente, é possível identificar duas classes distintas de cães e gatos, os quais possuem os mesmos atributos e métodos, mesmo que com valores diferentes. Porém essas classes pertencem a uma classe maior, os mamíferos, conforme demonstra a figura 3:

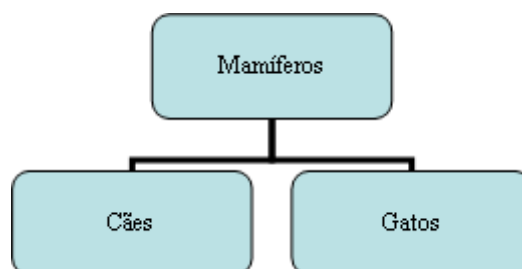


Figura 3 - Hierarquia de Classes

Para descrever o que é cão ou gato, pode ser omitido o que já foi apresentado na descrição de mamíferos. Para acrescentar uma nova classe de mamíferos, é preciso descrever apenas as características e comportamentos que os diferenciam dos outros mamíferos (CORREIA, 2006). As classes das quais as outras

dependem são chamadas de ancestrais, e as classes originadas a partir de outra são as descendentes (CORREIA, 2006). Assim, em cada descendente é preciso definir somente os atributos e métodos que o diferenciarão à descrição do ancestral.

2.2.5 Encapsulamento

O encapsulamento pode ser utilizado para diminuir o trabalho ao desenvolver um novo sistema, sendo esta a maior vantagem (COAD, 1991). Sendo realizada a identificação dos aspectos mais instanciáveis pelo analista, ao precisar realizar alterações dos requisitos, o processo será facilitado.

A vantagem do encapsulamento é que este disponibiliza o objeto com toda a sua funcionalidade sem necessidade de saber qual o seu funcionamento interno e como armazena os dados que são recuperados (CORREIA, 2006). Também permite a realização de alterações internas em um objeto sem afetar os outros componentes do sistema que utilizam este mesmo objeto.

2.2.6 Persistência

Persistência de objetos possui várias definições, sendo que sua definição consiste no tempo que um objeto permanece na memória (CORREIA, 2006). Desta forma, para tornar um objeto persistente, é necessário salvá-lo em algum meio de armazenamento, como por exemplo, um disco rígido.

Um exemplo de persistência pode ser analisado em um sistema de controle comercial que possua clientes como objetos. Estes objetos devem permanecer persistentes, pois estes precisam estar na “memória” do computador depois de digitados. Assim, os objetos clientes são considerados persistentes (CORREIA, 2006).

Um ponto importante é que a persistência de objeto não significa que o objeto será eterno. Então, quando for preciso retirar o cadastro de um cliente, este objeto deverá ser destruído, provocando a sua exclusão no cadastro de clientes do sistema.

2.2.7 Herança

Para exemplificar a herança pode-se utilizar a genética (CORREIA, 2006). O filho herda características genéticas dos pais e, por sua vez, repassa estas características aos seus filhos.

Pesquisando o significado de herança, esta possui as seguintes definições: “*s.f.* **1.** Aquilo que se herda. **2.** Conjunto dos bens que ficam ao herdeiro. **3.** Patrimônio deixado por alguém ao morrer; legado, posse. **4.** *Biol.* Aquilo que se transmite pelos genes ou com sangue; hereditariedade” (RIOS, 1999).

A partir destes conceitos, a herança também é aplicada em orientação a objetos, pois uma classe pode herdar características, métodos e atributos de outras. Da mesma forma uma classe pode repassar suas características para outras classes, fazendo com que as que receberam suas características se tornem suas herdeiras. De forma resumida, herança significa que todos os atributos programados no ancestral já estarão automaticamente presentes em seus descendentes sem precisar reescrevê-los.

São identificados dois tipos de herança, simples e múltipla. Denomina-se herança simples quando uma classe herda características de apenas uma superclasse (CORREIA, 2006). Herança múltipla é quando uma classe herda características de duas ou mais superclasses (CORREIA, 2006).

2.3 MODELO DE OBJETOS

Modelos são elaborados antes de construir algo por vários propósitos, servindo para diversos objetivos, como por exemplo, realizar testes em uma entidade física antes de definir uma forma para esta (RUMBAUGH, 1994). Assim, o modelo de objetos descreve a estrutura dos objetos de um sistema – sua identidade, seus relacionamentos com outros objetos, seus atributos e suas operações. Os objetos são as unidades em que o mundo é dividido – são as moléculas dos nossos modelos (RUMBAUGH, 1994).

O modelo de objetos é representado graficamente por diagramas de objetos contendo classes de objetos. Estas classes são organizadas em níveis hierárquicos compartilhando características comuns e são associadas a outras classes (RUMBAUGH, 1994).

Para que os objetos de *software* possam ser elaborados corretamente, de forma com que estes sigam um padrão, o uso de um modelo de objetos é a alternativa correta a ser seguida, com o intuito de definir as características desses objetos, como estes irão se comportar perante o sistema e os demais objetos, facilitando o entendimento das suas propriedades e vínculos. Dessa forma é possível, com objetos que contenham as mesmas estruturas, tornar o desenvolvimento mais organizado, diminuindo a chance de ocorrerem problemas relacionados a forma com que estes objetos foram modelados e permitindo uma representação mais próxima do mundo real.

2.4 BANCO DE DADOS ORIENTADO A OBJETOS

O conceito de orientação a objetos no desenvolvimento de sistemas também pode ser empregado na construção de um banco de dados. Assim, tem-se como resultado um banco de dados orientado a objeto (BDOO), o qual consiste em objetos que são interligados para refletir seus relacionamentos (BROOKSHEAR, 2003).

O surgimento deste modelo de banco de dados se deu pela necessidade de armazenar dados complexos, uma vez que estes precisavam ser quebrados em diversas tabelas, ou relações, para serem armazenados, sendo que para recuperar essas informações é preciso realizar um JOIN (junção) entre diversas tabelas (VASCONCELOS, 2013).

Com o uso da orientação a objetos, é possível realizar a modelagem de objetos de maneira que estes sejam representados de forma mais próxima ao mundo real. Um banco de dados orientado a objetos permite ainda que uma aplicação manipule objetos tanto persistentes ou não, uma vez que possibilita armazenar não apenas os atributos de um objeto, mas sim o objeto como um todo (VASCONCELOS, 2013).

Pode-se exemplificar o uso de um banco de dados orientado a objetos na seguinte situação: uma implementação de registros de funcionários poderia consistir em três classes, as quais seriam os tipos de objetos: FUNCIONARIO, CARGO e ATRIBUICAO. O objeto FUNCIONARIO estará ligado a uma coleção de objetos do tipo ATRIBUICAO, que representa as atribuições que aquele funcionário possui em particular. Por sua vez, cada objeto do tipo ATRIBUICAO está ligado a um objeto do tipo CARGO, que representa o cargo associado com aquela atribuição (BROOKSHEAR, 2003). Desta forma é possível identificar todas as atribuições de um funcionário a partir das ligações do objeto ao qual este é representado, e também será possível localizar todos os funcionários que possuem um cargo específico por meio da ligação com o objeto que representa o cargo (BROOKSHEAR, 2003).

As ligações entre os objetos em um banco de dados orientado a objetos geralmente serão mantidas pelo Sistema Gerenciador de Banco de Dados (SGBD), de maneira que os detalhes dessas ligações não sejam apresentados ao programador do software (BROOKSHEAR, 2003). Este é um conceito que diferencia um banco de dados orientado a objetos do modelo relacional, pois ao invés de ser utilizado o conceito de chave primária e secundária, estas são substituídas pelo identificador de objeto (OID – *Object Identifier*), o qual é controlado pelo próprio SGBD (VASCONCELOS, 2013). Sendo assim, ao ser acrescentado um novo objeto, a aplicação precisará apenas especificar os outros objetos aos quais ele se relacionará. O SGBD também terá que controlar a persistência das informações,

sendo que ao executar uma aplicação orientada a objetos, em geral os objetos usados são descartados quando o programa termina, sendo preciso que esses dados que são acrescentados sejam salvos após o término do programa que os criou (BROOKSHEAR, 2003).

De modo geral, pode-se concluir que o banco de dados orientado a objetos permite um melhor controle dos dados de uma aplicação utilizando-se do conceito de objetos como as classes que o sistema irá controlar, possibilitando ligações entre essas classes e facilitando a visualização desses dados.

3 MATERIAIS E MÉTODO UTILIZADO

A seguir serão descritas as ferramentas e o método que foram utilizados para a modelagem e desenvolvimento do protótipo do sistema:

3.1 VISUAL PARADIGM 8.2

As primeiras fases do processo de concepção de software foram feitas utilizando-se da ferramenta Visual Paradigm for UML, sendo que esta disponibiliza a execução da modelagem visual para todos os tipos de diagramas relacionados à UML, possuindo suporte para o gerenciamento dos Casos de Uso, bem como todos os diagramas relacionados à modelagem de sistemas (VISUAL PARADIGM, 2013). A figura 4 apresenta a tela de inicialização do Visual Paradigm:



Figura 4 - Tela de inicialização do Visual Paradigm

A versão 8.2 do *Visual Paradigm* possui os recursos básicos para a elaboração dos diagramas da UML na sua licença gratuita, sendo que para

corporações mais avançadas pode ser adquirida a versão completa do aplicativo, a qual proporciona várias outras funcionalidades. Para o desenvolvimento deste projeto, foi utilizada a versão gratuita, sendo que esta disponibilizar de um conjunto bom de diagramas e facilidade na utilização. O aplicativo está todo em inglês, o que pode dificultar um pouco o entendimento no início, porém com o uso torna-se mais prático.

3.2 LINGUAGEM DE PROGRAMAÇÃO JAVA

A linguagem Java foi implementada pela Sun Microsystems, em 1990, para ser executada nas mais diversas plataformas de hardware visando à Internet (FEDELI, 2003). Atualmente a linguagem pertence a Oracle.

A mesma foi desenvolvida para atender o princípio de portabilidade, uma vez que seria executada em qualquer plataforma independentemente do sistema operacional utilizado, não afetando o funcionamento do sistema operacional e dos aplicativos executados nele (FEDELI, 2003).

Ao compilar o código na linguagem Java, são gerados arquivos objetos chamados de *byte-codes*, e estes podem ser executados por meio de interpretadores desenvolvidos para cada tipo de plataforma (FEDELI, 2003).

Existem dois tipos de aplicativos Java. A própria aplicação Java é um deles e a outra é chamada de *Java Applets*. Os *Applets* tem função restrita para não interferir nos demais aplicativos que estiverem sendo executados pela plataforma (FEDELI, 2003).

Um outro diferencial da linguagem Java é o *JavaScript*, que são programas colocados em forma de código-fonte, os quais podem ser incluídos nos textos das páginas HTML. Estes programas escritos em *JavaScript* não precisam ser compilados e gerar *byte-codes*, pois estes são interpretados diretamente pelos *browsers* quando a página HTML for interpretada (FEDELI, 2003).

Para o desenvolvimento de aplicativos *desktop* a linguagem disponibiliza API's que auxiliam na criação de interfaces gráficas, sendo que em algumas IDE's

de desenvolvimento é possível a utilização de componentes gráficos de forma a otimizar a produtividade, com o recurso de clicar-arrastar, selecionando o componente desejado e posicionando este no local correto, sendo que a própria aplicação realiza o alinhamento de acordo com os demais componentes que foram ou serão utilizados.

3.3 IDE NETBEANS 7.1.1

Para o desenvolvimento do sistema descrito neste relatório, foram utilizados os conceitos de desenvolvimento da linguagem de programação Java. A IDE *NetBeans* foi escolhida por ser uma ferramenta que auxilia programadores a escrever, compilar, debugar e instalar aplicações. Esta foi elaborada com o intuito de simplificar o desenvolvimento e aumentar a produtividade, sendo que além de suportar a linguagem Java, o *NetBeans* proporciona suporte em C, C++, *PHP*, *XML* e linguagens *HTML*. A tela de inicialização do *NetBeans* é apresentada na figura 5:



Figura 5 - Tela de inicialização do NetBeans IDE

O *NetBeans* possui um grande conjunto de bibliotecas, módulos e *API's* (*Application Program Interface*, que são conjuntos de rotinas, protocolos e ferramentas para a construção de aplicativos de *software*), sendo que disponibiliza uma vasta documentação de forma bem organizada. Com estes recursos proporcionados, a IDE auxilia os desenvolvedores de maneira com que escrevam o *software* mais rapidamente. Atualmente está disponível em diversos idiomas, tornando-o cada vez mais popular, facilitando o acesso a iniciantes em programação e possibilitando o desenvolvimento de aplicativos multilíngue.

A IDE disponibiliza vários recursos de organização para o código, bem como meios de monitorar o funcionamento das rotinas em tempo de execução, bem como possibilita parar a execução do sistema em determinada linha de código, proporcionando o acompanhamento de como uma rotina funciona, permitindo assim a correção de possíveis erros de execução.

Para o desenvolvimento da parte visual do sistema, o *NetBeans* possui o recurso de clicar-arrastar componentes visuais, o que auxilia na criação das telas que o sistema possuirá, facilitando o tratamento das informações que serão apresentadas para o usuário e aumentando a produtividade da equipe de desenvolvimento.

3.4 BANCO DE DADOS NEODATIS ODB

É possível definir banco de dados como um conjunto de informações que são organizadas de forma lógica e física, possibilitando a manipulação desses dados.

Para manter e acessar um banco de dados é comumente utilizado um *software* denominado Sistema Gerenciador de Banco de Dados (*SGBD*), sendo que o termo banco de dados é utilizado em muitos casos como sinônimo de *SGBD*, por ser um *software* que incorpora as funções de definição, recuperação e alteração de dados de um banco de dados (HEUSER, 1999).

Para o desenvolvimento do protótipo do sistema, foi utilizada a ferramenta de gerenciamento de banco de dados *NeoDatis ODB*, a qual possui utilização simples,

que permite persistir os objetos de uma aplicação da maneira que eles foram elaborados na linguagem de programação nativa, sem ser necessário realizar qualquer tipo de conversão de dados. Por ser de utilização simples, este modelo de gerenciador de banco de dados pode ser utilizado em Java, como também em outras linguagens de programação, tais como *.Net*, *Google Android*, *Groovy* e *Scala* (NEODATIS, 2013).

A figura 6 apresenta a tela de configuração do arquivo de banco de dados que será manipulado pela ferramenta *NeoDatis* ODB:

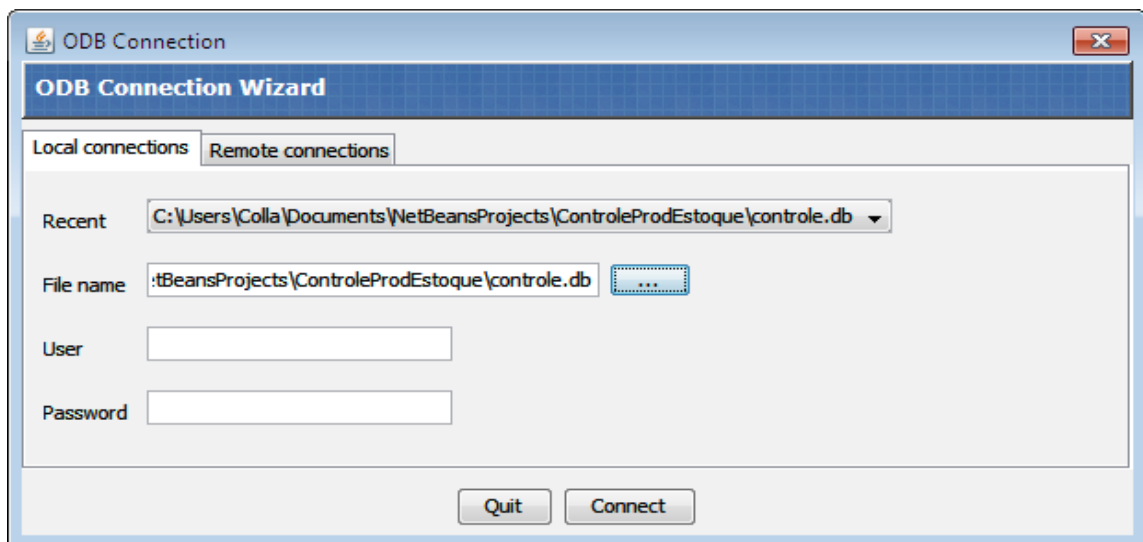


Figura 6 - Tela de configuração de conexão com BD do NeoDatis

3.5 METODOLOGIA ORIENTADA A OBJETOS

Para a elaboração deste trabalho, foi utilizada uma metodologia de desenvolvimento de software orientada a objetos que engloba as etapas de análise, projeto e implementação, a qual é chamada de OMT – *Object Modeling Technique* (ANTUNES).

Esta metodologia, com as demais metodologias orientadas a objetos, enxerga um software como sendo um conjunto de objetos de forma organizada que possuem

estruturas de dados e um comportamento específico. A abordagem da OMT utiliza os aspectos de Identidade, Classificação, Polimorfismo e Herança (ANTUNES).

Para o desenvolvimento de um software orientado a objetos com base na OMT, são executadas as etapas:

- Análise: Identificação do problema e elaboração do modelo para representar o que o sistema desejado deve fazer para solucionar a situação (ANTUNES);
- Desenho do sistema: definição da arquitetura do sistema e a plataforma onde o mesmo deverá rodar, “quebrando” o sistema em subsistemas para melhor avaliação de desempenho e a interface homem/máquina (ANTUNES);
- Desenho dos objetos: a partir do modelo obtido na etapa de análise, é feita a elaboração de um modelo de objetos, sendo que neste modelo constará detalhes de implementação, com base no que foi estabelecido na etapa de desenho do sistema como estratégia de implementação (ANTUNES);
- Implementação: o que foi projetado na etapa de desenho dos objetos como sendo classes e seus relacionamentos transformados em programas de computador, utilizando uma linguagem de programação e um determinado gerenciador de bases de dados, definindo o tipo de plataforma computacional que estes serão processados (ANTUNES).

A metodologia OMT utiliza três tipos de modelos para representar um sistema:

- Modelo de Objetos: é parecido com o clássico modelo de entidades-relacionamento, e descreve a estrutura dos objetos e seus respectivos relacionamentos em um sistema (ANTUNES);
- Modelo Dinâmico: descreve o ciclo de vida dos objetos do sistema, a evolução dos componentes do sistema ao longo do tempo (ANTUNES);

- Modelo Funcional: descreve os fluxos de entrada e saída de dados do sistema e os processos que transformam os dados que são informados ao sistema (entrada) em dados que serão retornados para o usuário (saída) (ANTUNES).

4 MODELAGEM DO SISTEMA

Este capítulo apresenta a modelagem realizada para o desenvolvimento do sistema para controle de produção e estoque de uma empresa de madeiras.

4.1 APRESENTAÇÃO DO SISTEMA

O gerenciamento das movimentações de estoque será de uma empresa de madeiras será a principal função do sistema. Para isso, o sistema irá controlar as rotinas de compra, venda e produção dos produtos, uma vez que a empresa adquire matérias-primas (toras, lâminas, resina, por exemplo) que serão utilizadas na produção interna (lâminas e compensados), registrando todas as movimentações de entradas e saídas para que seja possível identificar o saldo em estoque de cada especificação de produtos que a empresa manipula.

4.2 MODELAGEM DO SISTEMA

Para que seja possível o desenvolvimento de um sistema que atenda as reais necessidades do cliente, é preciso realizar uma modelagem correta das informações que serão controladas pelo sistema, demonstrando estes dados de forma a facilitar todo o processo de implementação, o que pode ser por meio de técnicas de modelagem de software, na utilização em metodologias de desenvolvimento ou em ferramentas de apoio.

4.2.1 Diagrama de Casos de Uso

Um diagrama de caso de uso exibe um conjunto de casos de uso e atores (um tipo especial de classe) e seus relacionamentos. Eles são utilizados para que se possa identificar como cada um dos elementos do sistema irá se comportar, deixando o sistema de uma forma mais fácil de compreender, pois apresentam uma visão externa de como esses elementos podem ser utilizados no contexto (BOOCH, 2005).

Abaixo são demonstrados os casos de uso do software de controle de produção e estoque com suas respectivas ligações entre os atores.

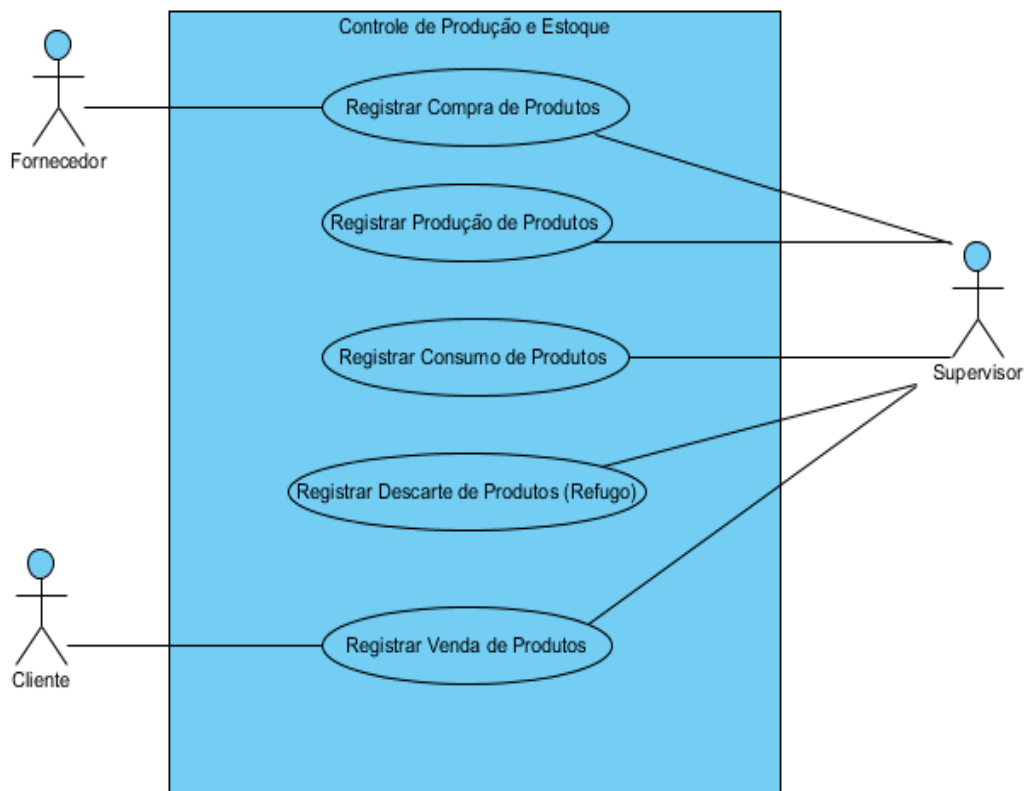


Figura 7 - Diagrama de Caso de Uso

Conforme apresentado na figura 7, foram identificados três atores, sendo que o Supervisor terá ligação com todos os casos de uso apresentados, podendo

realizar as movimentações de registro de compra, venda, produção, consumo e descarte dos produtos. O ator Cliente estará vinculado ao registro da venda para a identificação do destino do produto que foi produzido na empresa. Seguindo o mesmo raciocínio, o ator Fornecedor será relacionado com o registro de compra de produtos, possibilitando o controle da origem da matéria-prima que será empregada na produção.

Além dos casos de uso citados acima, também foram identificados os cadastros principais do sistema, denominados CRUD. A figura 8 abaixo apresenta estes requisitos:

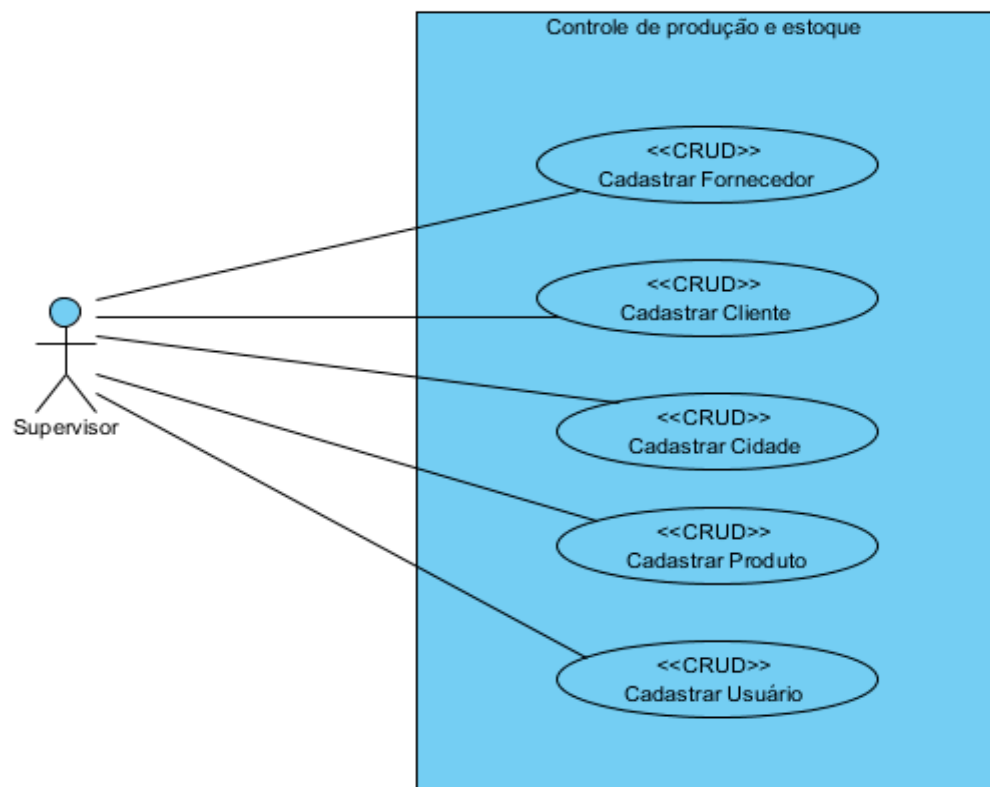


Figura 8 - Diagrama de Caso de Uso (CRUD)

4.2.2 Casos de Uso

A seguir serão apresentados os principais casos de uso do sistema, com os seus respectivos fluxos principais, os quais serão utilizados para o desenvolvimento.

4.2.1.1 Registrar Compra de Produtos

Os supervisores acessam o sistema com usuário e senha específicos.

No sistema, acessam o menu de cadastro de compra, sendo possível a realização de inclusão das compras de produtos que a empresa realiza, informando as quantidades adquiridas de cada especificação de produtos e de qual fornecedor foram adquiridas, e estas serão acrescentadas ao saldo de estoque para a utilização nos processos de produção.

4.2.1.2 Registrar Produção de Produtos

Os supervisores acessam o sistema o menu de cadastro de produção, e nesta tela serão mostradas todas as produções registradas no sistema, com as referidas datas e quantidades, sendo possível o vínculo dos produtos que foram utilizados para a produção cadastrada, sendo que a quantidade produzida será acrescentada ao saldo do produto produzido.

4.2.1.3 Registrar Consumo de Produtos

Os supervisores acessam o sistema o menu de cadastro de produção, sendo que para cada produção serão vinculados os produtos que foram consumidos com suas referidas quantidades, os quais devem estar cadastrados no sistema para que seja controlado o saldo em estoque, diminuindo o saldo dos produtos consumidos em cada produção.

4.2.1.4 Registrar Venda de Produtos

Os supervisores acessam o sistema o menu de cadastro de venda, nas quais serão especificados os produtos e suas quantidades, informando para qual cliente a venda será efetuada, realizando a baixa dos saldos do estoque dos produtos.

4.2.1.5 Cadastrar Fornecedores

O usuário acessará o sistema no menu Cadastros e selecionará a opção Fornecedores.

Informará os dados do fornecedor corretamente, e clicará no botão Salvar.

4.2.1.6 Cadastrar Clientes

O usuário acessará o sistema no menu Cadastros e selecionará a opção Clientes.

Informará os dados do cliente corretamente, e clicará no botão Salvar.

4.2.1.7 Cadastrar Produtos

O usuário acessará o sistema no menu Cadastros e selecionará a opção Produtos.

Informará os dados do produto corretamente, e clicará no botão Salvar.

4.2.1.8 Cadastrar Usuários

O usuário acessará o sistema no menu Cadastros e selecionará a opção Usuários.

Informará os dados do usuário corretamente, e clicará no botão Salvar.

4.2.1.9 Cadastrar Cidades

O usuário acessará o sistema no menu Cadastros e selecionará a opção Cidades.

Informará os dados da cidade corretamente, e clicará no botão Salvar.

4.2.3 Diagrama de Classes

A partir da identificação dos requisitos do sistema, foi possível identificar as classes e atributos, bem como os relacionamentos entre estas classes. A seguir será demonstrado o Diagrama de Classes que foi elaborado para o sistema.

A figura 9 apresenta a segunda parte do diagrama de classes, a qual demonstra as classes responsáveis pelos controles das movimentações dos produtos (Produção, Consumo, Compra, Venda e Descarte) e a identificação dos produtos através da classe principal **Produto**.

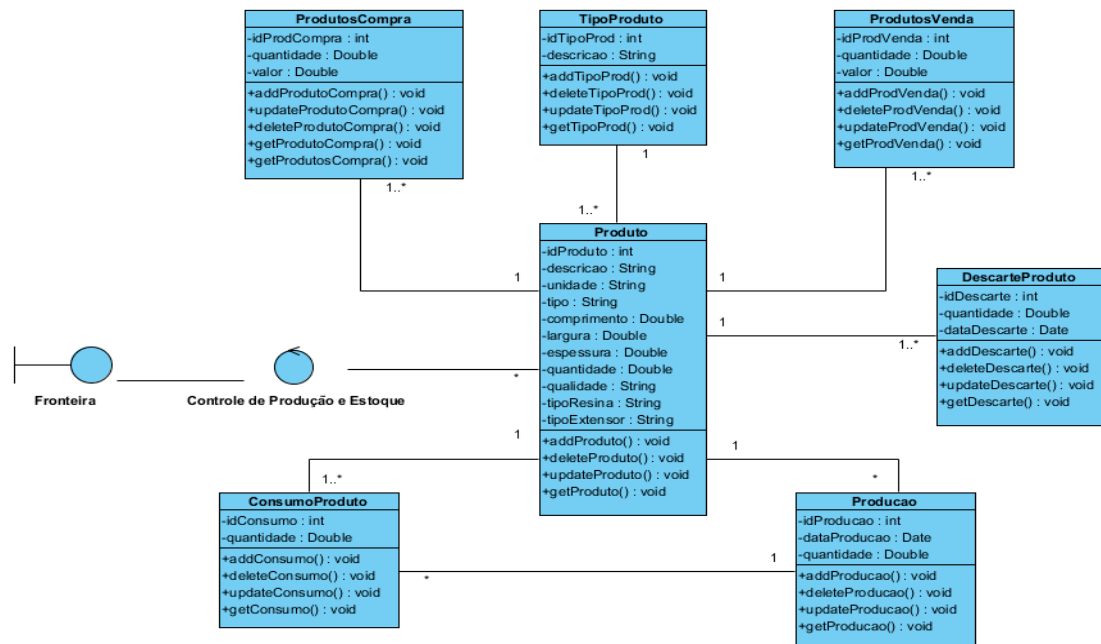


Figura 9 - Diagrama de Classes (parte I)

A figura 10 apresenta os relacionamentos com a classe principal **Fornecedor**, demonstrando as ligações referente à movimentação de compra de matéria-prima pela empresa.

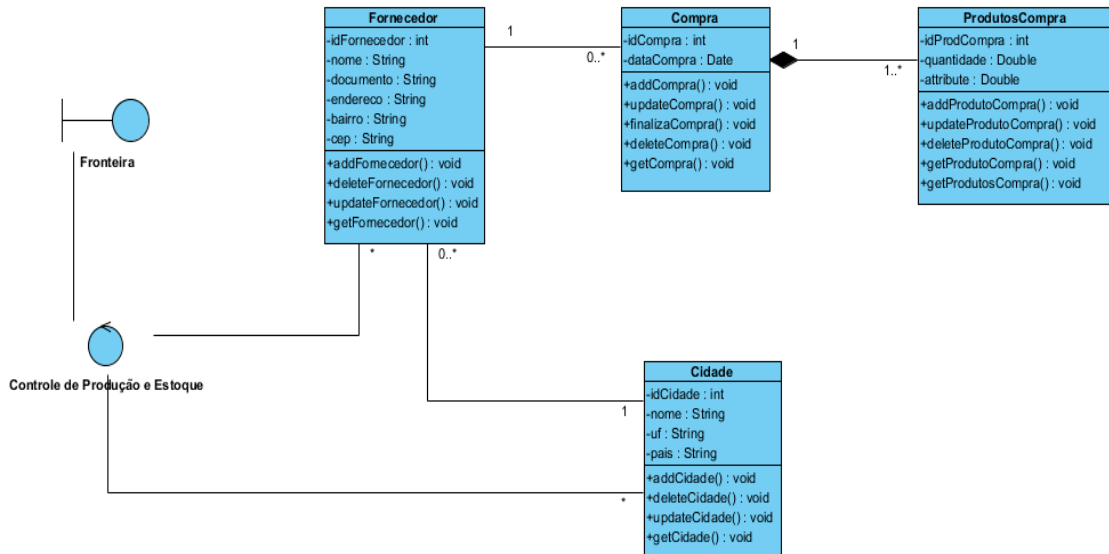


Figura 10 - Diagrama de Classes (parte II)

A classe **Controladora** tem por responsabilidade controlar os processos mais complexos do projeto, e a mesma está ligada à classe **Fronteira**, a qual representa as telas do sistema.

Por fim, na figura 11 são representadas as classes referentes à venda dos produtos. Nesta figura constam as classes completas **Usuario** e **Cliente**. A classe **Usuario** está ligada com a controladora apenas, pois esta representa o registro dos usuários que terão acesso às rotinas do sistema.

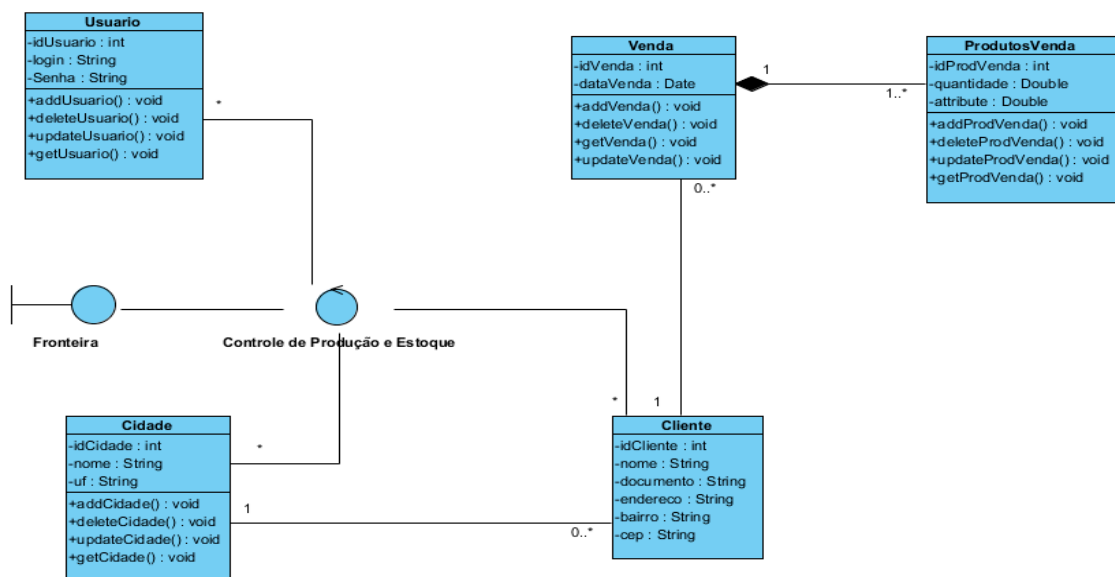


Figura 11 - Diagrama de Classes (parte III)

5 DESENVOLVIMENTO DO SISTEMA

Neste capítulo será apresentado como foi realizado o desenvolvimento do sistema, o processo de criação do banco de dados, bem como as classes configuradas para a comunicação com a base de dados NeoDatis ODB através da ferramenta NetBeans IDE.

5.1 ORGANIZAÇÃO DAS CLASSES DO PROJETO

Como foi utilizado o conceito de orientação a objetos, cada objeto identificado na modelagem do sistema foi implementado como uma classe no projeto, as quais foram organizadas em pacotes, cada uma segundo as suas funções. Foi também empregada a arquitetura de desenvolvimento em camadas, organizando a aplicação em três unidades de distribuição (RICARTE):

- Camada de interface com o usuário, onde encontram-se as telas do sistema;
- Camada de negócios, com os processos lógicos de controle da aplicação;
- Camada de dados, responsável por gerenciar os dados.

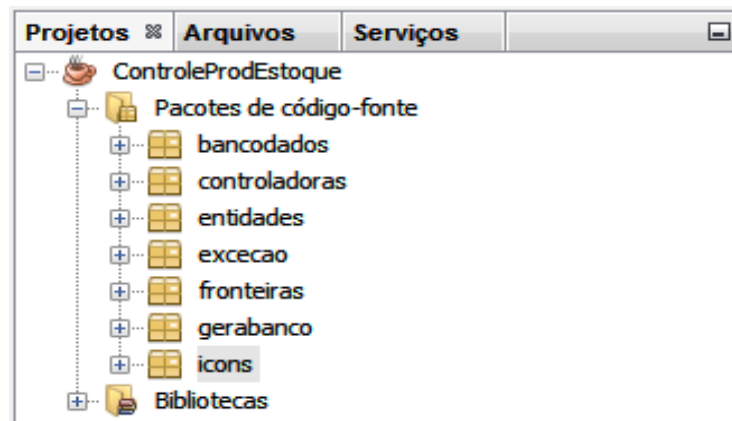


Figura 12 - Pacotes do Projeto

A organização das classes do projeto gerou os pacotes de códigos fonte demonstrados na figura 12. O pacote “bancodados” possui as classes que irão desempenhar a função de conexão com o banco de dados para armazenar as informações digitadas no sistema. As classes do pacote “controladoras” possuem as rotinas para listagem dos registros e para salvar e excluir os mesmos. Já no pacote “entidades” foram codificadas as classes que identificam os campos que serão utilizados em cada cadastro, contendo as funções para atribuir e buscar valores de cada campo (os chamados *getters* e *setters*).

No pacote “excecao” foram implementadas classes as quais realizam o tratamento de exceções que podem ocorrer em determinado momento do sistema. As classes do pacote “fronteiras” são as que possuem as telas que serão apresentadas ao usuário, contendo os campos que serão informados os dados e a partir daí os mesmos serão gravados no banco de dados pela comunicação entre as classes fronteiras com as controladoras e entidades.

A classe pertencente ao pacote “gerabanco” foi codificada para ser executada uma única vez, antes de ser executado o sistema como um todo. Nela constam todos os campos de todas as tabelas que o sistema irá utilizar para armazenamento das informações. A partir da execução dessa classe, o banco de dados será criado da forma com que os campos estão dispostos no código, uma vez que esta classe possui vínculo com a classe *Conexao.java* do pacote “bancodados” e também com todas as classes do pacote “entidades”, pois em cada entidade são definidos os campos de informações pertencentes, sendo estes replicados na criação da base de dados.

Por fim o pacote “icons” possui as imagens que serão utilizadas como ilustração no sistema nas telas de cadastro.

5.2 PADRÃO DE NOMENCLATURAS DAS VARIÁVEIS

Para um melhor controle do projeto, foi seguido um padrão para a definição da nomenclatura das variáveis.

Os pacotes de códigos-fonte foram nomeados com a descrição toda minúscula, como no exemplo “bancodados”, sem separador entre as palavras. As classes do projeto foram denominadas com a primeira letra de cada palavra em maiúsculo e as demais em minúsculo, sendo que ao possuir mais de uma palavra, as mesmas não terão separador (“ProducaoControle”).

Os componentes visuais do projeto foram nomeados de acordo com o componente utilizado (tf – JTextField; cb – JComboBox; jt – Jtable; bt – JButton; pf – JpasswordField; lb - JLabel) no início do nome das variáveis, e após uma descrição da variável em si (por exemplo: tfIdProduto; cbTipoProd; jtLista; btSalvar; pfSenha; lbSaldoEstoque).

Desta forma foi facilitada a identificação de cada componente no sistema e a sua real função.

5.3 CRIAÇÃO DO BANCO DE DADOS

Para a criação da base de dados do sistema, foi escolhida uma ferramenta gerenciadora integrada à IDE de desenvolvimento NetBeans, a qual denomina-se NeoDatis ODB. A mesma difere-se das demais ferramentas de manipulação e criação de banco de dados para softwares por possibilitar a criação de registros a partir de código interno da própria linguagem de desenvolvimento, não sendo necessária a utilização de comandos SQL para cadastrar as tabelas que serão controladas pelo sistema.

Com essa possibilidade de gerenciar as informações referentes à base de dados através de código interno, foi elaborada uma classe para realizar esse controle, a qual foi denominada “CriaBanco.java”, e será descrita posteriormente.

5.3.1 Classe Conexao.java

O quadro 4 apresenta o código gerado para a realização da conexão com o banco de dados, sendo que nesta classe é realizada a instanciação dos objetos “instance” e “odb”, que farão a comunicação entre o sistema e a base de dados, sendo que nesta classe foram importadas as bibliotecas do NeoDatis.

```
package bancodados;

import org.neodatis.odb.ODB;
import org.neodatis.odb.ODFactory;

public class Conexao {
    // Atributo que informará o estado da conexão
    // Qualquer objeto desta classe terá o mesmo valor
    // para o atributo

    private static Conexao instance;
    private ODB odb;

    private Conexao() {
        odb = ODFactory.open("controle.db");
    }

    public static Conexao getInstance() {
        if (instance == null) {
            instance = new Conexao();
        }
        return instance;
    }

    public ODB getConexao() {
        return odb;
    }
}
```

```
public void fechaConexao() {  
    odb.close();  
}
```

Quadro 4 - Classe Conexao.java

5.3.2 Classe Fabrica.java

No quadro 5 a seguir é apresentado o código da classe Fabrica.java:

```
package bancodados;  
  
import java.util.ArrayList;  
import org.neodatis.odb.Objects;  
  
public class Fabrica<K extends Object> {  
  
    public ArrayList<K> listar(Class tipo) {  
        Objects<K> result = Conexao.getInstance().getConexao()  
            .getObjects(tipo);  
        return new ArrayList<>(result);  
    }  
    public void salvar(K objeto) {  
        Conexao.getInstance().getConexao().store(objeto);  
        Conexao.getInstance().getConexao().commit();  
    }  
    public void excluir(ArrayList<K> lista) {  
        for (K obj : lista) {  
            Conexao.getInstance().getConexao().delete(obj);  
        }  
        Conexao.getInstance().getConexao().commit();  
    }  
}
```

Quadro 5 - Classe Fabrica.java

A classe `Fabrica.java` é responsável pelo controle dos objetos que serão manipulados pelas classes controladoras afim de fazer a comunicação desses dados com o banco de dados. No quadro 5 é apresentado a parte do código onde são informadas as bibliotecas utilizadas pela classe. Além da biblioteca `ArrayList` utilizada para realizar o controle da lista de registros, também é utilizada a biblioteca de objetos do `NeoDatis`.

Para o funcionamento da classe `Fabrica.java`, a mesma precisa de um parâmetro, sendo este o objeto que será utilizado para a manipulação das rotinas de listar, salvar e excluir, sendo que com base nesse parâmetro, a classe irá realizar a comunicação com o registro no banco de dados correto.

5.3.3 Classe `CriaBanco.java`

Com a classe responsável pela conexão com o banco de dados criada, foi elaborada a classe que realizará a criação dos registros no banco de dados. A classe "`CriaBanco.java`" utiliza-se da classe `Conexao.java` para que seja possível informar quais tabelas o banco de dados possuirá, e quais dados serão manipulados por cada tabela. O quadro 6 a seguir apresenta o código gerado para instanciar os registros dos objetos da classe `Producao` no banco de dados do sistema:

```
Producao producao = new Producao();
producao.setIdProducao(1);
producao.setProduto(produto);
producao.setDataProducao(new Date());
producao.setQuantidade(1.0);
Conexao.getInstance().getConexao().store(producao);
```

Quadro 6 - `CriaBanco.java` - Produção

O quadro apresentado acima mostra a forma como são registrados os dados no banco de dados com a utilização do `NeoDatis`. Cada registro é declarado conforme a descrição dos dados que irão ser manipulados. Como no exemplo

acima, são definidos quais campos cada objeto será composto. Assim, a classe `Producao` terá um campo código (“`idProducao`”), um campo produto (“`produto`”, o qual será outro objeto que terá relacionamento com a classe `Producao`), um campo `DataProducao` (que espera um valor do tipo `Date`) e um campo `Quantidade` (o qual espera um valor do tipo `Double`). Por fim, é chamado o método `getInstance` da classe `Conexao`, passando por parâmetro o objeto “`produção`”. A definição dos dados que serão registrados para cada tabela são definidos nas classes do pacote `Entidades`.

5.3.4 Entidade `Producao.java`

Como descrito nas seções anteriores, as classes do pacote “`entidades`” foram codificadas contendo os dados que serão utilizados para o armazenamento das informações de cada registro do sistema. Sendo assim, os campos que são definidos nestas classes serão os utilizados para a criação das tabelas do banco de dados.

Os quadros a seguir apresentam o código da classe `Producao.java`:

```
package entidades;

import java.util.ArrayList;
import java.util.Date;

public class Producao {
    private int idProducao;
    private Date dataProducao;
    private Produto produto;
    private double quantidade;
    private ArrayList<ProdutosConsumo> consumo;
    private boolean status;

    public Producao() {
```

```
        consumo = new ArrayList();
    }

    public Producao(int idProducao, Date dataProducao, Produto produto,
double quantidade) {
        this.idProducao = idProducao;
        this.dataProducao = dataProducao;
        this.produto = produto;
        this.quantidade = quantidade;
        this.consumo = new ArrayList();
    }
}
```

Quadro 7 - Entidade Producao.java

No quadro 7 é possível verificar a parte do código da classe Producao.java onde são declaradas as variáveis que a entidade Producao possuirá, as quais serão: int idProducao (armazena o código da produção), Date dataProducao (armazena a data da produção), Produto produto (armazena o produto vinculado à produção), Double quantidade (armazena a quantidade produzida) e ArrayList<ProdutosProducao> itensProducao (armazena os produtos que foram utilizados para a produção). A partir da declaração das variáveis, é instanciado o método construtor dessas variáveis, passando-as como parâmetro.

Com as variáveis devidamente informadas, é gerado o código para a manipulação dessas variáveis, que são os métodos *getters* e *setters*, os quais serão os responsáveis por permitir a consulta e a atribuição de valores a essas variáveis pelas classes que utilizarão essas informações. Esta parte do código não foi apresentada por ser codificada com base em um padrão da própria IDE, a qual gera os métodos *getters* e *setters* automaticamente a partir dos atributos que são declarados na classe.

5.3.5 Controladora ProducaoControle.java

Uma vez as entidades codificadas, foram elaboradas as controladoras do sistema, as quais terão as funções de gerenciar os fluxos de salvar, excluir e listar os registros do sistema.

A seguir é apresentado o código-fonte da controladora ProducaoControle.java:

```
package controladoras;

import bancodados.Fabrica;
import entidades.Producao;
import excecao.ObrigatorioExcecao;
import java.util.ArrayList;

public class ProducaoControle {

    private static Fabrica<Producao> fabrica = new Fabrica<>();

    public static ArrayList<Producao> listarProducao() {
        return fabrica.listar(Producao.class);
    }

    public static void salvarProducao(Producao producao) throws
    ObrigatorioExcecao {
        fabrica.salvar(producao);
    }

    public static void excluirProducao(ArrayList<Producao> con) {
        fabrica.excluir(con);
    }

    public static int primeiraProducao() {
        int maior = 0;
        ArrayList<Producao> aproducao = fabrica.listar(Producao.class);
        for (Producao producao : aproducao) {
            if (producao.getIdProducao() > maior){
```

```
        maior = producao.getIdProducao();  
    }  
}  
return maior + 1;  
}
```

Quadro 8 - Controladora ProducaoControle.java

As classes controladoras do sistema utilizarão a classe Fabrica.java do pacote “bancodados”, a qual instanciará um objeto referente ao registro da controladora. No caso do quadro 8, é instanciado um objeto Fabrica<Producao>, o qual informa o tipo do objeto que será manipulado por essa controladora. Na classe ProducaoControle.java utilizará também a entidade Producao.java, pois serão os campos declarados pela entidade que serão controlados e manipulados.

Logo após é apresentado o código referente às funções salvarProducao (a qual é passada por parâmetro a variável producao) e excluirProducao (sendo passado por parâmetro um ArrayList da entidade produção).

5.3.6 Fronteira Login.java

Estando as classes responsáveis pelo banco de dados, entidades e controladoras codificadas, foi possível realizar a implementação da parte visual do projeto, as quais foram salvas no pacote chamado “fronteiras”. Como foram definidas regras para o acesso ao sistema, por padrão a primeira tela que será apresentada será a tela de validação de login, a qual foi denominada Login.java, sendo que esta é apresentada na figura 13:

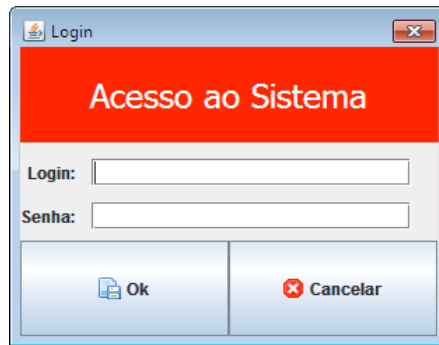


Figura 13 - Tela de Login do sistema

Nesta tela o usuário do sistema irá informar o login e a senha cadastrados para poder realizar o acesso às demais funcionalidades do sistema. Este controle é preciso para que somente usuários com acesso permitido possam utilizar do sistema e consultar os dados que nele estão registrados.

Será feita uma validação no momento em que o usuário informar o login e senha e clicar no botão "Ok", sendo consultado no banco de dados se o registro informado existe. Caso o mesmo não esteja cadastrado, será apresentada a mensagem da Figura 14, e se for informado um login e senha válidos, o sistema irá apresentar ao usuário a tela principal.

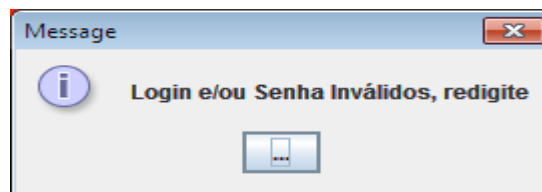


Figura 14 - Mensagem de validação do login

5.3.7 Fronteira MenuPrincipal.java

Após o usuário informar o login e senha corretamente, a próxima tela que será apresentada é a referente a classe MenuPrincipal.java, a qual apresentará todas as rotinas que o sistema disponibiliza.

A figura 15 apresenta a tela do menu principal do sistema, a qual possui os menus de Cadastro, Movimentação, Relatórios e Ajuda, possibilitando o acesso as demais rotinas do sistema.

Ao acessar a tela principal do sistema, serão apresentadas as abas de Cadastro, onde serão feitos os registros dos usuários do sistema, clientes, fornecedores, produtos, unidade de medida e cidades, a aba de Movimentação, onde o usuário acessará as opções de compra, venda e produção, e a aba de Relatórios, onde serão acessados os relatórios do sistema, além da aba Ajuda, que apresenta informações sobre a finalidade do desenvolvimento do sistema.

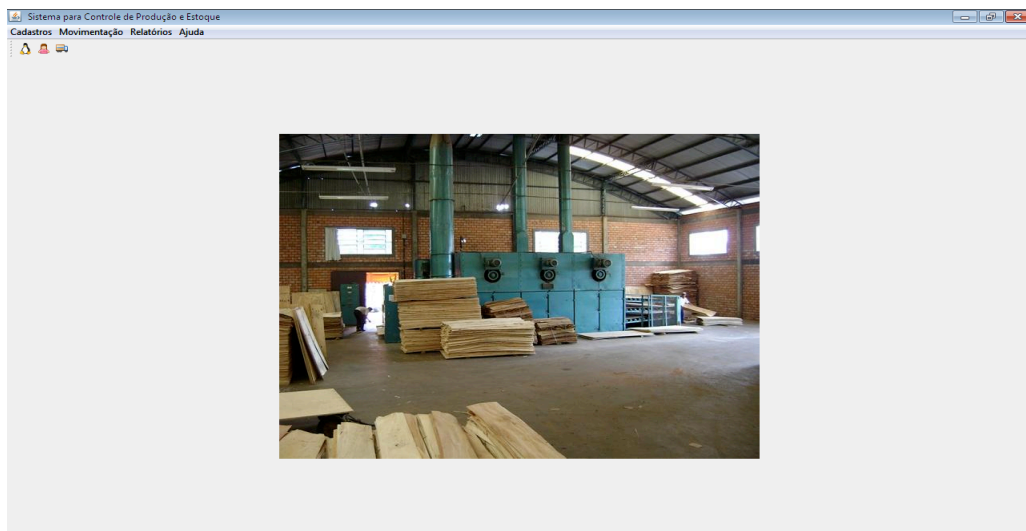


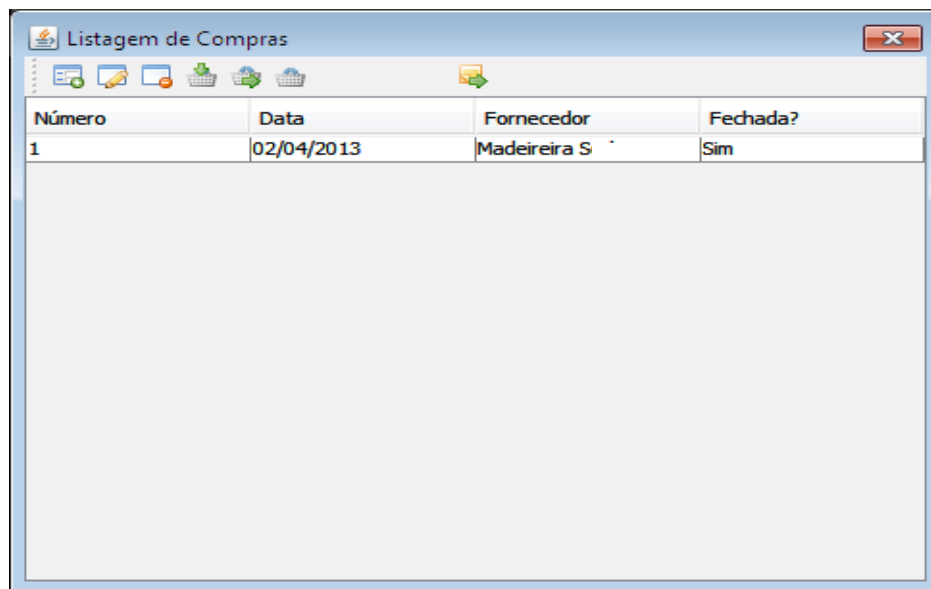
Figura 15 - Menu principal do sistema

5.3.8 Fronteira Compra.java

Para demonstrar um exemplo de tratamento das fronteiras do sistema, será apresentada a fronteira Compra.java.


Ao acessar o menu Movimentação => Compra, o sistema irá direcionar o usuário para outra fronteira, a qual apresentará a listagem das compras cadastradas, a fronteira ListaCompra.java, apresentada na figura 16.

Em todas as telas de listagem que o sistema apresenta, estas possuem alguns botões com as funcionalidades de inclusão, alteração e exclusão de registros, e também como no exemplo da tela de Listagem de Compras, também é possível incluir novo produto no registro de uma compra cadastrada, no botão Inserir Item na Compra, sendo que para isso ser possível, a referida compra não pode estar fechada, opção esta que se dá pelo botão Fechar Nota. Ao fechar uma compra, a mesma não poderá mais ser alterada. Também há a opção para visualizar todos os itens da compra, no botão Mostrar Itens da Compra.



Número	Data	Fornecedor	Fechada?
1	02/04/2013	Madeira S	Sim

Figura 16 - Tela de Listagem de Compras

Ao clicar no botão Inserir Novo Registro , o sistema executa o seguinte código:

```
private void btInserirActionPerformed(java.awt.event.ActionEvent evt) {
    JDialog cadastro = new CadastroCompra(null, true);
    cadastro.setVisible(true);
    lista();
}
```

Quadro 9 - Inserir novo registro

O código apresentado no quadro 9 faz com que o sistema seja direcionado para a tela de Cadastro de Compra, apresentada na figura 17 a seguir:

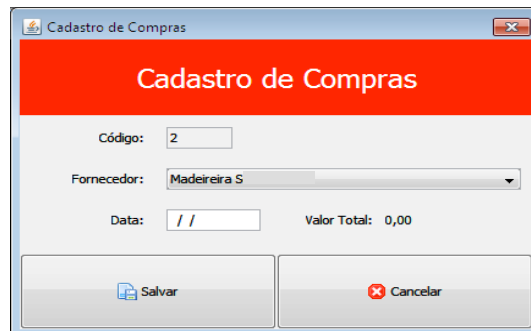



Figura 17 - Tela de Cadastro de Compras

Ao ser apresentada, a tela de cadastro de compras trará no campo Código o próximo código disponível para registro, seguindo uma ordem sequencial, para que não existam dois registros de compra com o mesmo código. O campo Fornecedor trará a lista dos fornecedores que estão cadastrados no sistema, sendo que este já estará informado com o primeiro registro cadastrado. Também terá o campo Data, sendo que neste será informada a data em que a compra foi realizada, e o campo Valor Total, que apresentará o valor total da referida compra.

Ao salvar essa compra, a mesma será apresentada na tela de listagem, e assim estará disponível para que sejam informados os produtos que foram comprados, conforme apresentada na figura 16.

Na tela de Listagem de Compras, ao selecionar um registro da lista e clicar no botão de Alterar Registro Selecionado , o sistema irá executar o seguinte código:

```
private void btAlterarActionPerformed(java.awt.event.ActionEvent evt)
{
    if (jtLista.getSelectedRowCount() == 1) {
        Compra aux = (Compra)lista.get(jtLista.getSelectedRow());
        if (aux.isStatus()){
            javax.swing.JOptionPane.showMessageDialog(this,
                "Nota já fechada, não será alterada");
        }else{
            CadastroCompra cadastro = new CadastroCompra(
                null, true, lista.get(jtLista.getSelectedRow()));
            cadastro.setVisible(true);
        }
    }
}
```

```
    }  
    lista();  
} else {  
    JOptionPane.showMessageDialog(this, "Selecione apenas uma  
compra da lista.");  
}  
}
```

Quadro 10 - Código de alteração de registro

O código demonstrado no quadro 10 inicia fazendo o teste se o usuário selecionou apenas um registro, caso esta opção não esteja de acordo, será apresentada a mensagem da figura 18 ao usuário:

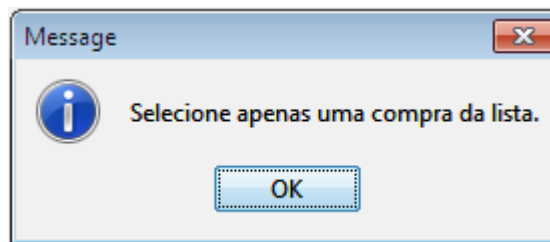


Figura 18 - Mensagem para seleção de compra da lista

Ao instanciar um objeto Compra “aux” é possível fazer o teste se a compra já está fechada ou não, se a mesma não estiver fechada, será apresentada a tela de cadastro da compra selecionada, com os dados cadastrados inicialmente, possibilitando a alteração dos dados. Caso a compra esteja fechada, será então apresentada a mensagem da figura 19:

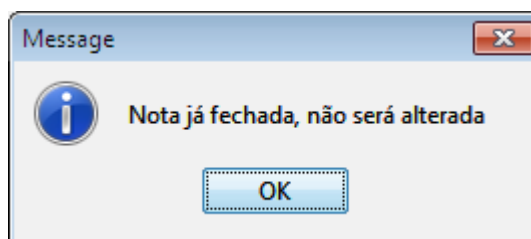



Figura 19 - Mensagem de Nota Fechada

Também pela tela de Listagem de Compras é possível realizar a exclusão de uma compra, clicando no botão Excluir registro selecionado , será executado o código apresentado no quadro 11:

```
private void btExcluirActionPerformed(java.awt.event.ActionEvent evt) {  
    if (jtLista.getSelectedRowCount() > 0) {  
        ArrayList<Compra> paraExcluir = new ArrayList<>();  
        for (int index : jtLista.getSelectedRows()) {  
            if (lista.get(index).isStatus()) {  
                javax.swing.JOptionPane.showMessageDialog(this,  
                    "Nota já fechada, não será excluída");  
            } else {  
                paraExcluir.add(lista.get(index));  
            }  
        }  
        int opc;  
        opc = JOptionPane.showConfirmDialog(this, "Confirma exclusão  
de registros?");  
        if (opc == 0) {  
            CompraControle.excluirCompra(paraExcluir);  
            lista();  
        }  
    } else {  
        JOptionPane.showMessageDialog(this,  
            "Selecione pelo menos uma venda para excluir.");  
    }  
}
```

Quadro 11 - Código de exclusão de registro

Primeiramente, a rotina realiza a validação se há alguma compra selecionada, caso não, será apresentada uma mensagem, conforme apresentada na figura 20:

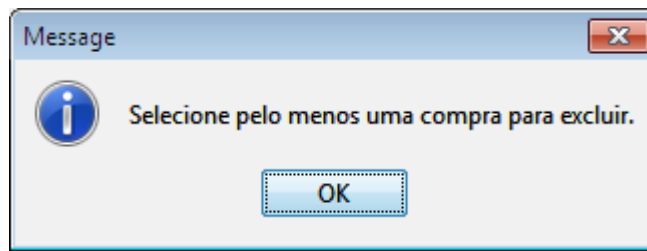


Figura 20 - Mensagem de seleção de registro para exclusão

Caso haja um registro selecionado, será então apresentada a mensagem apresentada na figura 21 para confirmação da exclusão do registro:

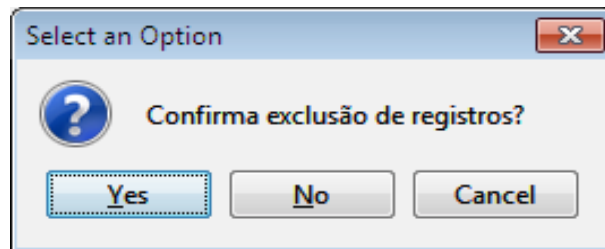



Figura 21 - Mensagem de confirmação de exclusão

Ao confirmar a exclusão, o registro selecionado será excluído, chamando a rotina de exclusão de compra, localizada na classe CompraControle.

Caso seja clicado no botão Inserir Item na Compra , a aplicação executa a seguinte rotina:

```
private void
btInserirItemCompraActionPerformed(java.awt.event.ActionEvent evt) {
    if (jtLista.getSelectedRowCount() == 1) {
        Compra aux = (Compra)lista.get(jtLista.getSelectedRow());
        if (aux.isStatus()){
            javax.swing.JOptionPane.showMessageDialog(this,
                "Nota já fechada, não podem ser incluídos
itens");
        }else{
            CadastroItemCompra cadastro = new CadastroItemCompra(
                null, true, lista.get(jtLista.getSelectedRow()));
            cadastro.setVisible(true);
        }
    }
}
```

```
    }  
    lista();  
} else {  
    JOptionPane.showMessageDialog(this, "Selecione uma compra.");  
}  
}
```

Quadro 12 - Código para Inserir Item

Neste código constante no quadro 12 também é feita a validação se há algum registro selecionado, caso não, será apresentada a mensagem demonstrada na figura 22:

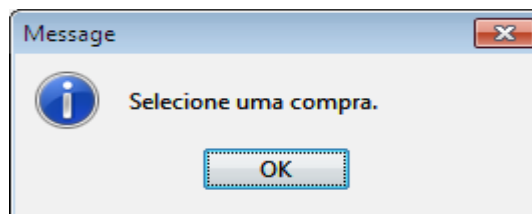


Figura 22 - Mensagem para selecionar uma compra

Caso haja um registro selecionado, será então instanciado um objeto Compra “aux”, para realizar a validação se a compra já está fechada ou não, pois se a mesma estiver com o status fechada, não será mais possível incluir novos itens. Caso seja possível, será então apresentada ao usuário a tela de cadastro de novo item da compra na Figura 23:

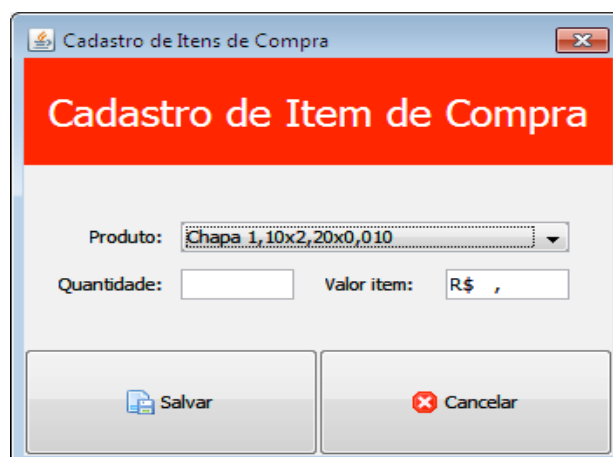



Figura 23 - Tela de Cadastro de Itens da Compra

Desta forma será feito o registro de todos os produtos que foram adquiridos na compra, e após isso, para que as quantidades sejam efetivadas em estoque, será preciso na tela de Listagem de Compras, selecionar a referida compra e clicar na opção Fechar a Nota , finalizando a compra e realizando a atualização do saldo dos produtos que foram informados na compra. Será apresentada uma mensagem para confirmação do fechamento da nota, conforme figura 24:

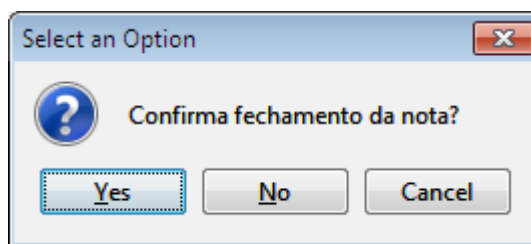



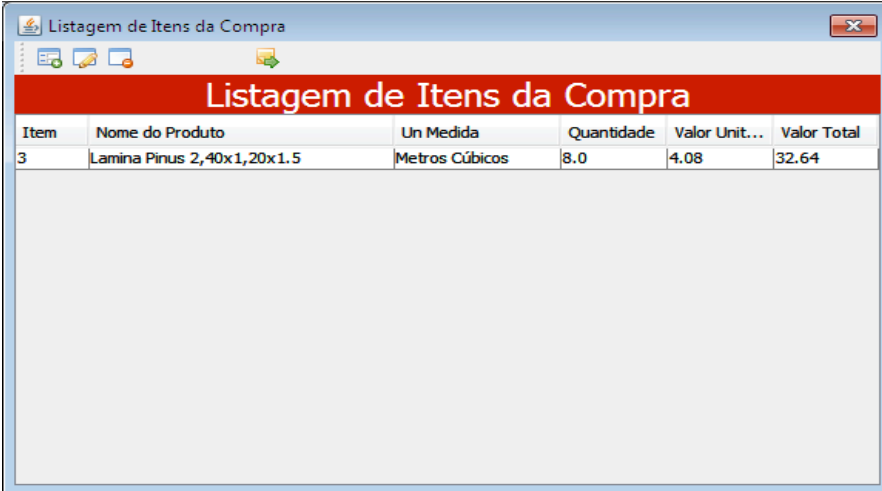
Figura 24 - Mensagem de confirmação para fechar nota

Será também possível através da tela de Listagem de Compras selecionar uma compra cadastrada e visualizar os itens que já foram inclusos na mesma, através do botão Mostrar Itens da Compra , o qual irá executar o código apresentado no quadro 13:

```
private void
btMostrarItemCompraActionPerformed(java.awt.event.ActionEvent evt) {
    if (jtLista.getSelectedRowCount() == 1) {
        ListaItemCompra listaitemcompra = new ListaItemCompra(
            null, true, lista.get(jtLista.getSelectedRow()));
        listaitemcompra.setVisible(true);
        lista();
    } else {
        JOptionPane.showMessageDialog(this, "Selecione apenas uma
compra da lista.");
    }
}
```

Quadro 13 - Código para Visualizar registros cadastrados

Será feita novamente a validação se há uma compra selecionada, e caso o usuário selecione uma compra, será então criado um objeto do tipo `ListItemCompra`, apresentando a lista de todos os itens da referida compra, conforme apresentado na Figura 25:



Item	Nome do Produto	Un Medida	Quantidade	Valor Unit...	Valor Total
3	Lamina Pinus 2,40x1,20x1.5	Metros Cúbicos	8.0	4.08	32.64

Figura 25 - Tela de Listagem de Itens da Compra

As demais telas de cadastro de movimentações do sistema (Produção e Venda) seguem uma estrutura parecida com a rotina de compra demonstrada acima, com as opções de inclusão, alteração e exclusão dos registros, bem como a listagem dos itens de cada produção ou venda registrados, as quais só farão a movimentação dos saldos dos produtos em estoque ao realizar a finalização do registro, através da opção Fechar Nota.

6 CONCLUSÃO

Este estudo teórico e prático realizado durante o desenvolvimento do trabalho proposto atendeu aos requisitos que foram levantados com base na modelagem do sistema.

Os objetivos destacados foram alcançados, sendo realizado o levantamento dos requisitos do sistema e a modelagem dos mesmos com o auxílio da ferramenta *Visual Paradigm*.

Com a análise feita, foi desenvolvido o *software* para controle das movimentações de estoque para uma empresa de produção de madeiras utilizando-se dos conceitos de orientação a objetos para a implementação da parte visual e armazenamento de dados, com o uso da ferramenta *NetBeans IDE 7.1* para o desenvolvimento do código-fonte e da ferramenta *NeoDatis ODB* para a criação e manutenção da base de dados, aprimorando o conhecimento das mesmas.

Com o sistema concluído, foi possível verificar que suas rotinas atendem às funcionalidades que foram identificadas como sendo necessárias para que o controle dos processos internos de produção e estoque de uma empresa de madeiras seja otimizado.

O *software* para gerenciamento de estoque para madeiras será aprimorado em possíveis trabalhos futuros, tendo em vista a realização de novas implementações e/ou correções.

REFERÊNCIAS

ANTUNES, Dante Carlos; DE OLIVEIRA, Paulo Antonio Fuck; MENDES, Rogério da Fonseca. **Orientação a Objetos.** Disponível em: <<http://www.batebyte.pr.gov.br/modules/conteudo/conteudo.php?conteudo=1027>>. Acesso em 08/05/2013.

BOOCH, Grady. RUMBAUGH, James. JACOBSON, Ivar. **UML : guia do usuário** – Rio de Janeiro : Elsevier, 2005 – 9ª Reimpressão.

BROOKSHEAR, J. Gleen. **Ciência da Computação – Uma visão abrangente.** Porto Alegre: Bookman, 2003.

COAD, Peter; YOURDON, Edward. **Análise Baseada em Objetos.** Rio de Janeiro: Campus, 1991.

CORREIA, Carlos Henrique; TAFNER, Malcon Anderson. **Análise Orientada a Objetos.** Florianópolis: Visual Books, 2006.

FEDELI, Ricardo Daniel; POLLONI, Eurico Giulio Franco; PERES, Fernando Eduardo. **Introdução à ciência da computação.** São Paulo: Pioneira Thomson Learning, 2003.

GONÇALVES, Edson. **Desenvolvendo Relatórios Profissionais com iReport para Netbeans IDE.** Ciência Moderna, 2009.

HEUSER, Carlos Alberto. **Projeto de Bando de Dados.** Porto Alegre, SagraLuzzato, 1999.

NEODATIS. **NeoDatis Objetc Database.** Disponível em <<http://neodatis.wikidot.com/>>. Acesso em 04/04/2013.

NETBEANS IDE. Disponível em <<http://netbeans.org/features/index.html>>. Acesso em 04/04/2013.

RICARTE, Ivan L. M. **Arquitetura de três camadas.** Disponível em: <<http://www.dca.fee.unicamp.br/cursos/PooJava/servlets/trescamadas.html>>. Acesso em 10/09/2013.

RIOS, Dermival Ribeiro. **Mini dicionário escolar da língua portuguesa.** São Paulo: DCL, 1999.

RUMBAUGH, James; BLAHA, Michael; PREMERLANI, William; EDDY, Frederick; LORENSEN, William. **Modelagem e Projetos Baseados em Objetos.** Rio de Janeiro: Campus, 1994.

VASCONCELOS, Rafael Oliveira; PINHEIRO, Daniel Ramon Silva; SOUZA, Danilo Santos; SILVA, Fábio Soares. **Comparativo entre Banco de Dados Orientado a Objetos (BDOO) e Bancos de Dados Objeto Relacional (BDOR)**. Disponível em <<http://rafaeloliveirav.wordpress.com/category/artigos/>>. Acesso em 13/04/2013.

VISUAL PARADIGM. Disponível em <<http://www.visual-paradigm.com>>. Acesso em 04/04/2013.