CURSO TÉCNICO DE INFORMÁTICA

Tielle da Silva Alexandre

MÓDULO VII

Ministério da Educação

GOVERNO FEDERAL
BRASIL
PAÍS RICO É PAÍS SEM POBREZA

Tielle da Silva Alexandre
Módulo: VII

Projeto Final I

Disciplina do Eixo de Disciplinas do Currículo do
Curso Técnico de Informática CEFET/RJ UnED NI

Edição: CEFET/RJ UnED NI – COORDENAÇÃO DE INFORMÁTICA

Local: Estrada de Adrianópolis, 1317 – Santa Rita, Nova Iguaçu - RJ

Editora: CEFET/RJ

Ano de Publicação: 2016

CEFET/RJ

rede e-Tec Brasil

Presidente da República
Dilma Rousseff

Ministro da Educação
Cid Ferreira Gomes

Secretário de Educação Profissional e Tecnológica
Eliezer Moreira Pacheco

Professora – organizadora
André Alexandre Guimarães Couto

Diretor Geral do CEFET/RJ
Carlos Henrique Figueiredo Alves

Diretora de Ensino
Gisele Maria Ribeiro Vieira

Coordenadora da Educação à Distância no CEFET/RJ
Maria Esther Provenzano

Coordenador Geral do e-Tec no CEFET/RJ
Mauro Godinho Gonçalves

Coordenador Geral Adjunto do e-Tec no CEFET/RJ
Alexandre Martinez dos Santos

Coordenadora do Curso de Informática e-Tec no CEFET/RJ
Rosana Soares Gomes Costa

Coordenador de Polo Nova Iguaçu do e-Tec no CEFET/RJ
Francisco Eduardo Cirto

Coordenador de Tutoria do e-Tec no CEFET/RJ
Unapetinga Hélio Bomfim Vieira

Professora Pesquisadora do e-Tec no CEFET/RJ
Lucia Helena Dias Mendes

Design Instrucional
Luciana Ponce Leon Montenegro de Morais Castro

Ficha catalográfica elaborada pela Biblioteca Central do CEFET/RJ

Curso técnico de Informática, módulo VII: Projeto Final I: disciplina do Eixo de Disciplinas Transversais ao Currículo do Curso Técnico de Informática CEFET-RJ / Priscilla Leite Corrêa e Castro (org.). Rio de Janeiro: CEFET/RJ, – 2016.

Apresentação do e-Tec Brasil

Prezado estudante,

Bem vindo ao e-Tec Brasil!

Você faz parte de uma rede nacional pública de ensino, a Escola Técnica Aberta do Brasil, instituída pelo Decreto n° 6.301, de 12 de dezembro de 2007, com o objetivo de democratizar o acesso ao ensino técnico público, na modalidade a distância. O programa é resultado de uma parceria entre o Ministério da Educação, por meio das Secretarias de Educação Profissional e Tecnológica (SETEC), as universidades e escolas técnicas estaduais e federais.

A educação à distância no nosso país, de dimensões continentais e grande diversidade regional e cultural, longe de distanciar, aproxima as pessoas ao garantir acesso à educação de qualidade, e promover o fortalecimento da formação de jovens moradores de regiões distantes, geograficamente ou economicamente, dos grandes centros.

O e-Tec Brasil leva os cursos técnicos a locais distantes das instituições de ensino e para a periferia das grandes cidades, incentivando os jovens a concluir o ensino médio. Os cursos são ofertados pelas instituições públicas de ensino e o atendimento ao estudante é realizado em escolas-polo integrantes das redes públicas municipais e estaduais.

O Ministério da Educação, as instituições públicas de ensino técnico, seus servidores técnicos e professores acreditam que uma educação profissional qualificada – integradora do ensino médio e educação técnica, - é capaz de promover o cidadão com capacidades para produzir, mas também com autonomia diante das diferentes dimensões da realidade cultural, social, familiar, esportiva, política e ética.

Nós acreditamos em você!

Desejamos sucesso na sua formação profissional!

Ministério da Educação

Janeiro de 2010

Nosso contato

etecbrasil@mes.gov.br

Indicação de ícones

Curiosidades: indica informações interessantes que enriquecem o assunto.

Interrogação: indica perguntas frequentes do aluno em relação ao tema e respostas às mesmas.

Você sabia? : oferece novas informações e notícias recentes relacionadas ao tema estudado.

Lembrete: enfatiza algum ponto importante sobre o assunto.

Tome nota 1: espaço dedicado às anotações do aluno.

Tome nota 2: espaço também dedicado às anotações do aluno.

Mãos a obra: apresenta atividades em diferentes níveis de aprendizagem para que o estudante possa realizá-las e conferir o seu domínio do tema estudado.

Bibliografia: apresenta a bibliografia da apostila.

SUMÁRIO

COM A PALAVRA, O PROFESSOR...

Caros (as) alunos (as):

O Projeto Final é uma disciplina que visa à revisão teórica dos conhecimentos adquiridos em disciplinas anteriores. Nesta disciplina será revisado todo o conteúdo necessário para a construção de um *software* por completo. Ao final de cada aula, o aluno irá realizar atividades práticas a fim reforçar o conteúdo. Portanto, essa disciplina possibilita uma visão geral do conhecimento necessário para se criar um *software.*

Para que o aluno consiga acompanhar o desenvolvimento da disciplina de Projeto Final, o aluno deve obedecer ao cronograma de estudo proposto, bem como fazer e entregar todas as atividades práticas para a correção. Portanto, a organização e o comprometimento são essenciais nesta empreitada.

Ao término das três últimas aulas da disciplina de Projeto Final, o aluno terá desenvolvido de forma incremental um pequeno *software* contendo métodos para cadastrar, consultar, alterar e excluir (CRUD) registros do banco de dados segundo o padrão de desenvolvimento Model-View-Controller - MVC.

O organizador.

Apresentação da Disciplina

Módulo VII – Projeto Final I

Carga Horária: 30 Horas

Espera-se que o (a) cursista desenvolva as seguintes competências:

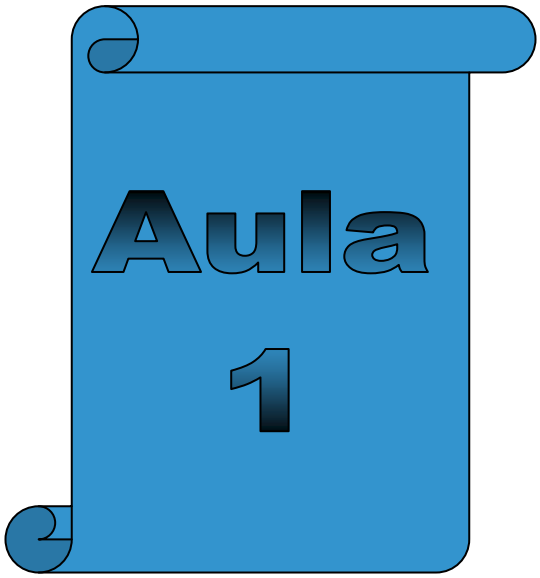
- Compreenda os esquemas de um projeto de banco de dados, bem como a aplicação de todos eles;
- Desenvolver interfaces gráficas;
- Entender e aplicar o padrão *Model-View-Controller* –MVC;
- Desenvolver uma pequena aplicação com as operações básicas de inserção, atualização, deleção e de consulta de dados, utilizando o Sistema Gerenciador de Banco de Dados MySQL e a linguagem de programação *Java Standard Edition* - SE.

Projeto instrucional

Disciplina: Projeto Final I (30 horas)

Ementa: Modelagem e definição de um banco de dados. Desenvolvimento de interfaces Gráficas. Utilização do padrão MVC.

AULA	OBJETIVOS DE APRENDIZAGEM	MATERIAIS	CARGA HORÁRIA 30 (horas)
1- Projeto Final.	Conhecer a disciplina de Projeto Final e a definição do modelo conceitual de banco de dados.	impresso	4
2- Modelo lógico e físico de um banco de dados.	Conhecer a ferramenta MySQL e criar um modelo lógico e físico de um banco de dados.	impresso	5
3 – Criação de interfaces gráficas.	Projetar e criar a interfaces gráficas.	impresso	4
4 – Entendendo o padrão *Model-View-Controller* - MVC.	Entender os conceitos do padrão MVC e a implementação de uma classe DAO.	impresso	5
5 – Implementação do padrão MVC – Método Cadastrar	Implementação do método cadastrar utilizando o modelo MVC.	impresso	4
6 – Implementação do padrão MVC – Método Consultar	Implementação do método consultar utilizando o modelo MVC.	impresso	4
7 – Implementação do padrão MVC – Método Excluir e Alterar	Implementação do método excluir e alterar utilizando o modelo MVC.	impresso	4



Projeto Final

Olá, caro estudante!

Saudações Cefetianas!

Antes de iniciarmos nosso estudo sobre Tecnologia, reflita sobre essa ideia:

"Tudo aquilo que não enfrentamos em vida acaba se tornando o nosso destino."

Carl Jung

1. Projeto Final

O Projeto Final é uma disciplina prática que visa à aplicação do conhecimento teórico adquirido em disciplinas anteriores. Em cada aula, uma breve revisão será realizada com a utilização de exemplos e o aluno realizará atividades práticas sobre o conteúdo revisado. A disciplina de Projeto Final está relacionada com as disciplinas de Banco de Dados e Tópicos Avançados III, portanto, em caso de dúvida ao conteúdo abordado é recomendo que aluno revise as apostilas dessas disciplinas.

Nas duas últimas aulas da disciplina de Projeto Final, o aluno irá desenvolver um pequeno *software* passando pelo desenvolvimento de um projeto de banco de dados, a definição dos requisitos do sistema, a implementação da interface gráfica e aplicação do padrão de desenvolvimento *Model-View-Controller* - MVC.

Nesta aula, serão revisados alguns conceitos básicos da área de banco de dados que são relevantes para a compreensão de um projeto de banco de dados. O aluno revisará o esquema conceitual de banco de dados; o Modelo Entidade-Relacionamento (MER) e a sua representação gráfica; os requisitos funcionais e não funcionais e as regras de negócios.

Segundo Heuser (1998), um modelo de banco de dados é uma descrição dos tipos de informações que estão sendo armazenadas em um banco de dados. Um modelo de dados é descrito em três níveis diferentes de abstração de dados: o modelo conceitual, lógico e físico.

O modelo conceitual é o modelo mais abstrato por não depender de implementação em um Sistema Gerenciador de Banco de Dados (SGBD). Portanto, o objetivo desse modelo é representar de forma abstrata o banco de dados sem fazer qualquer menção com a tecnologia usada na implementação do modelo de banco de dados.

O modelo conceitual envolve uma descrição do ambiente em que o *software* será implantado. Nessa descrição, são identificados as regras de negócio e os requisitos funcionais e os requisitos não funcionais que *software* deve atender. Essa descrição é chamada de mini-mundo. Como

exemplo, a descrição do mini-mundo da Locadora Cão Sentado é apresentado a seguir:

Mini-Mundo da Locadora Cão Sentado

A Locadora Cão Sentado necessita de um sistema para gerenciar as locações de filmes. O sistema deve ser implementado com o Sistema Gerenciador de Banco de Dados, o *MySQL* e com a linguagem de programação *Java SE*. De cada filme é armazenado o título, o gênero, a duração (em minutos) e a sinopse. Um filme possui um ou mais atores e um mesmo ator pode atuar em um ou mais filmes. De cada ator é pertinente armazenar o nome e nacionalidade. Para um determinado filme, a Locadora Cão Sentado pode possuir vários itens, em diferentes mídias (DVD ou *Bluray*). Um item possui um tipo de mídia. O sistema deve permitir aos clientes consultarem os itens para locação.

Os clientes locam itens. Apenas clientes maiores de idade podem ser cadastrados e o sistema deve gerar um número de inscrição único para cada cliente cadastrado. Um cliente é cadastrado com o CPF, o nome, o endereço, a data de nascimento e o *e-mail* e a locadora possui diversos clientes. Um cliente pode fazer a locação de muitos itens de filmes, mas um item só pode ser locado por apenas um cliente. Em relação à locação de um item, o sistema deve armazenar a data de locação e de devolução e o preço. O cliente que não devolver o item alugado até a data de devolução estipulada não pode realizar novas locações.

O modelo entidade relacionamento (MER) é gerado a partir da descrição do mini-mundo. Sendo assim, os componentes desse modelo são identificados a partir de uma leitura minuciosa da descrição do mini-mundo. Os componentes do MER são: *entidade, atributo, relacionamento, generalização/especialização* e a entidade *associativa*. O MER é representado graficamente, através de um diagrama entidade-relacionamento (DER).

A notação gráfica de Chen (1977) pode ser considerada como um

padrão para modelagem conceitual. A seguir, é apresentada uma síntese sobre os conceitos do MER e a notação gráfica segundo Chen (1977) para representar esses conceitos no diagrama entidade-relacionamento.

- Entidade

A entidade representa um conjunto de objetos sobre os quais se deseja guardar informações (Heuser, 1998). Uma entidade pode representar objetos concretos (ex. cliente, aluno) e objetos abstratos (ex. endereço) da realidade modelada. Por convenção, o nome de uma entidade deve estar no singular.

No digrama entidade-relacionamento, uma entidade é representada por retângulo, conforme ilustra a Figura 1.

Figura 1. Representação gráfica de uma entidade.

A entidade cliente irá manter informações sobre um conjunto de clientes no banco de dados. Cada seja necessário se referir a um cliente específico, esse é chamado de ocorrência de uma entidade.

- Atributo

Um conjunto de características está associado a cada ocorrência de uma entidade. Por exemplo, cada ocorrência de uma entidade cliente possui um nome, um CPF, um *e-mail* e o endereço referentes a um determinado cliente. Os atributos são representados graficamente, conforme mostra a Figura 2.

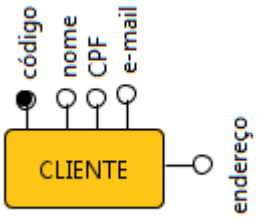


Figura 2. Representação gráfica dos atributos de uma entidade.

Cada identidade deve possuir um identificador que serve para distinguir de forma única, uma ocorrência de cliente das demais ocorrências. No diagrama entidade-relacionamento, o identificador é representado por um círculo preto. No caso da entidade cliente da Figura 2, o identificador é o código. Assim, podemos afirmar que cada cliente possui um código que identifica cada cliente, unicamente.

- Relacionamentos

Um relacionamento representa uma relação entre dois tipos de entidades onde estas precisam compartilhar as informações. O relacionamento deve acontecer com no mínimo duas entidades (relacionamento binário). Existem três tipos de relacionamentos:

1. um-para-um: uma ocorrência da entidade X pode se relacionar com no máximo uma ocorrência da entidade Y e uma ocorrência da entidade Y pode se relacionar com no máximo uma ocorrência da entidade X. No digrama entidade-relacionamento, esse tipo de relacionamento é representado conforme mostra a Figura 3.

Figura 3. Representação gráfica de um relacionamento 1:1.

2. um-para-muitos: uma ocorrência da entidade X pode ser relacionar com mais de uma ocorrência da entidade Y e uma ocorrência Y pode se relacionar com no máximo uma ocorrência da entidade X. No digrama entidade-relacionamento, esse tipo de relacionamento é representado conforme mostra a Figura 4.

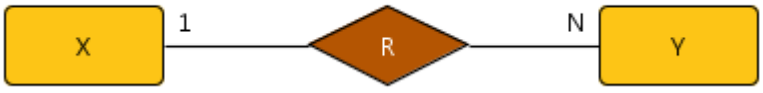


Figura 4. Representação gráfica de um relacionamento 1:N.

3. muitos-para-muitos: uma ocorrência da entidade X pode ser relacionar com mais de uma ocorrência da entidade Y e uma ocorrência Y pode se relacionar com mais de uma ocorrência da entidade X. No digrama entidade-relacionamento, esse tipo de relacionamento é representado conforme mostra a Figura 5.

.

Figura 5. Representação gráfica de um relacionamento N:N.

- Generalização/Especialização

Esse conceito permite a existência de superclasses e de subclasses. Uma entidade superclasse possui características genéricas que são aplicadas a qualquer entidade de um determinado tipo. Uma entidade subclasse possui características específicas, além de herdar as características de uma entidade superclasse.

Por exemplo, a entidade superclasse *cliente* possui como características o código, o nome e o telefone. A subclasse F*ísica* possui como característica específica o CPF e a subclasse *Jurídica* possui o CNPJ como característica específica. As subclasses *Físicas e Jurídicas* herdam as características o código, o nome e telefone da superclasse *cliente*. No digrama entidade-relacionamento, esse tipo de relacionamento é representado conforme mostra a Figura 6.

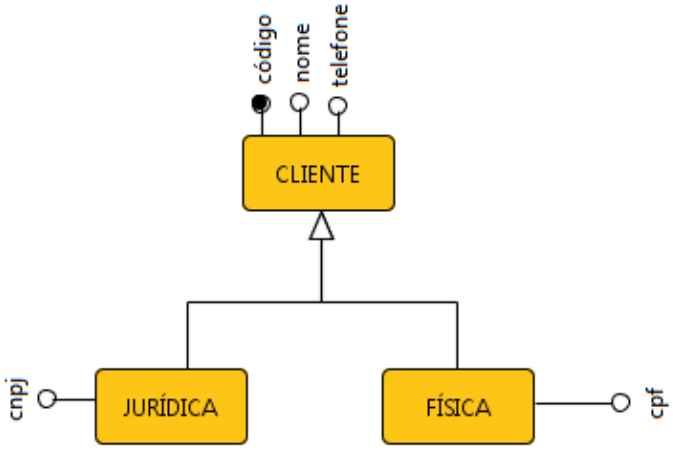


Figura 6. Representação gráfica de generalização/especialização.

- Entidade associativa

Esse conceito permite que entidades relacionadas possam se relacionar com outra entidade. Segundo Heuser (1998), uma entidade associativa nada mais é que a redefinição de um relacionamento, que passa ser tratado como se fosse também uma entidade. No digrama entidade-relacionamento, esse tipo de relacionamento é representado conforme mostra a Figura 7.

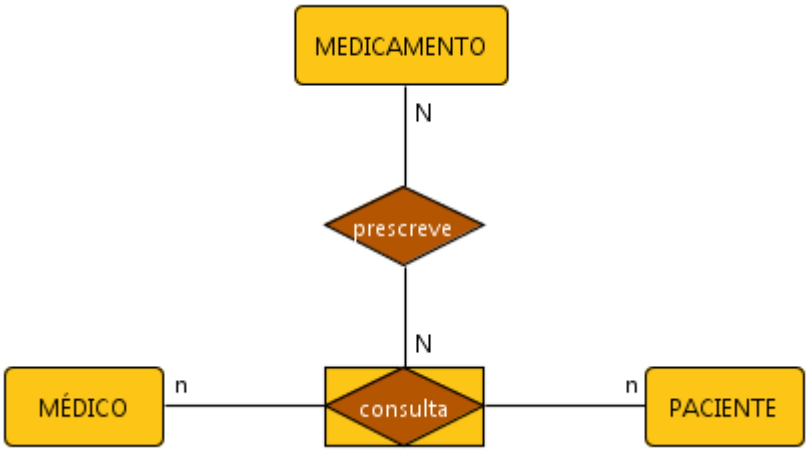


Figura 7. Representação gráfica de uma entidade associativa.

Note que um médico atende o paciente e nessa consulta pode ou não haver uma prescrição de um medicamento.

Através de uma leitura minuciosa do mini-mundo, os componentes do MER são identificados. Para facilitar esse processo, recomenda-se sublinhar as entidades e circular os atributos encontrados. Aplicando essa recomendação no mini-mundo da Locadora Cão Sentado é obtido o seguinte resultado:

Mini-Mundo da Locadora Cão Sentado

A Locadora Cão Sentado necessita de um sistema para gerenciar as locações de filmes. O sistema deve ser implementado com o Sistema Gerenciador de Banco de Dados, o *MySQL* e com a linguagem de programação *Java SE*. De cada filme é armazenado o título, o gênero, a duração (em minutos) e a sinopse. Um filme possui um ou mais atores e um mesmo ator pode atuar em um ou mais filmes. De cada ator é pertinente armazenar o nome e nacionalidade. Para um determinado filme, a Locadora Cão Sentado pode possuir vários itens, em diferentes mídias (DVD ou *Bluray*). Um item possui um tipo de mídia. O sistema deve permitir aos clientes consultarem os itens para locação.

Os clientes locam itens. Apenas clientes maiores de idade podem ser cadastrados e o sistema deve gerar um número de inscrição único para cada cliente cadastrado. Um cliente é cadastrado com o CPF, nome, endereço, a data de nascimento e o *e-mail* e a locadora possui diversos clientes. Um cliente pode fazer a locação de muitos itens de filmes, mas um item só pode ser locado por apenas um cliente. Em relação à locação de um item, o sistema deve armazenar a data de locação e de devolução e o preço. O cliente que não devolver o item alugado até a data de devolução estipulada não pode realizar novas locações.

Após a leitura do mini-mundo da Locadora Cão Sentado foi identificado quatro entidades: filme, ator, item e cliente. Os atributos da entidade filme são: título, gênero, duração e a sinopse. Os atributos da entidade ator são: nome e a nacionalidade. Os atributos da entidade item são: tipo, data de locação e de devolução e o preço. Os atributos da entidade cliente são: CPF, nome, *e-mail,* a data de nascimento e o endereço. A representação do diagrama entidade-relacionamento é ilustrada na Figura 8.

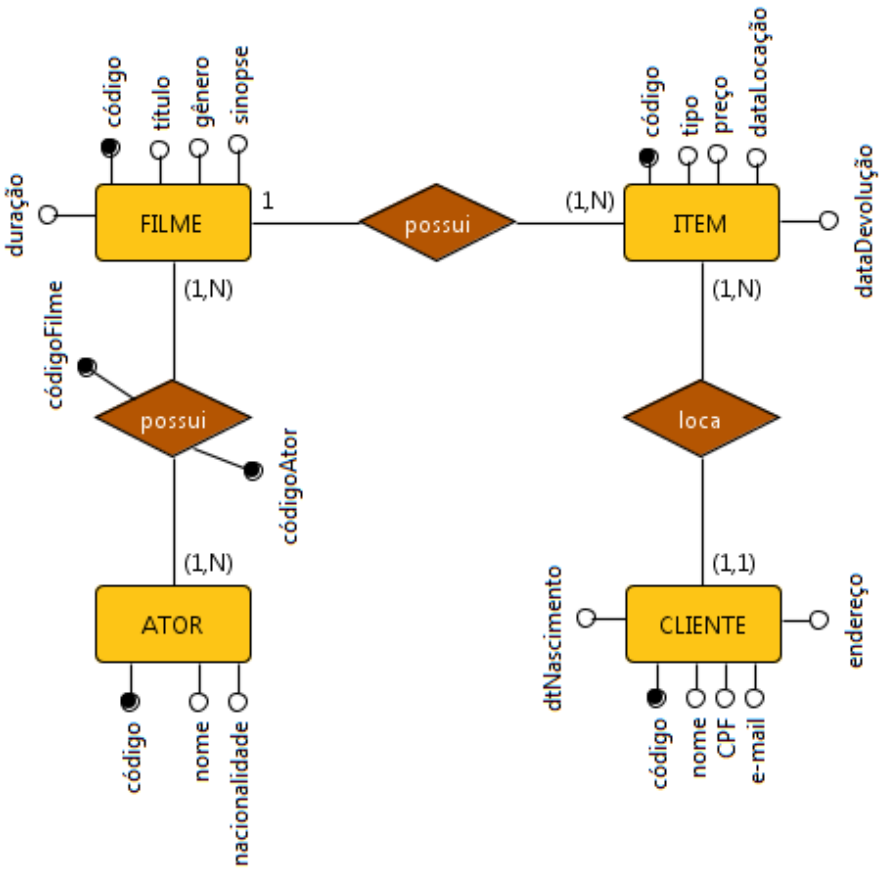


Figura 8. Diagrama entidade-relacionamento da Locadora Cão Sentado.

Os requisitos funcionais são as funções que o sistema deve desempenhar. No sistema da Locadora Cão Sentado, por exemplo, pode-se citar como requisitos funcionais:

- O sistema deve permitir o cadastro de um cliente;
- O sistema deve permitir que o cliente realize uma consulta dos itens para a locação;
- O sistema deve permitir a alteração de um item.

Os requisitos não funcionais estão relacionados ao uso da aplicação, envolvendo especificamente a parte técnica. Esse tipo de requisito abrange aspectos como desempenho, implementação, integração entre outros. No sistema da Locadora Cão Sentado, pode-se citar como requisito não

funcional:

- O sistema deve ser implementado com o Sistema Gerenciador de Banco de Dados, o *MySQL* e com a linguagem de programação *Java SE*.

As regras de negócio são declarações que definem ou restringem o funcionamento de uma operação. Logo se não houver uma regra restringindo, tudo será permitido. No sistema da Locadora Cão Sentado, pode-se citar como regra de negócio:

- Apenas clientes maiores de idade podem ser cadastrados e o sistema deve gerar um número de inscrição único para cada cliente cadastrado;
- O cliente que não devolver o item alugado até a data de devolução estipulada não pode realizar novas locações.

Aula 2

Modelo lógico e físico de um banco de dados

2. Modelo lógico e físico de um banco de dados.

Nesta aula, serão demonstrados os conceitos necessários para realizar o mapeamento do modelo conceitual em um modelo lógico de banco de dados. O modelo lógico será derivado a partir de um determinado diagrama entidade-relacionamento. Por fim, o modelo físico será criado e implementado com base no modelo lógico. Além disso, será feita uma breve revisão da linguagem *Structured Query Language* –SQL para a definição e manipulação de dados.

O modelo lógico é a segunda etapa do projeto de banco de dados. Segundo Heuser (1998), um modelo lógico é uma descrição de um banco de dados no nível de abstração visto pelo usuário do Sistema Gerenciador de Banco de Dados - SGBD. Sendo assim, o modelo lógico depende do tipo de SGBD e por isso, é dito que o modelo lógico é atrelado a uma tecnologia. Nesta apostila serão considerados apenas os modelos lógicos referentes SGBD relacional.

Esse modelo define quais tabelas com as suas respectivas colunas deveram ser implementadas. Além disso, há a adequação da nomenclatura das tabelas e colunas e a definição de chaves primárias e estrangeiras. Portanto, o modelo lógico representa uma estrutura de dados.

Para realizar o mapeamento do modelo conceitual para o modelo lógico será utilizado como exemplo, o diagrama entidade-relacionamento de uma loja ilustrado na Figura 9.

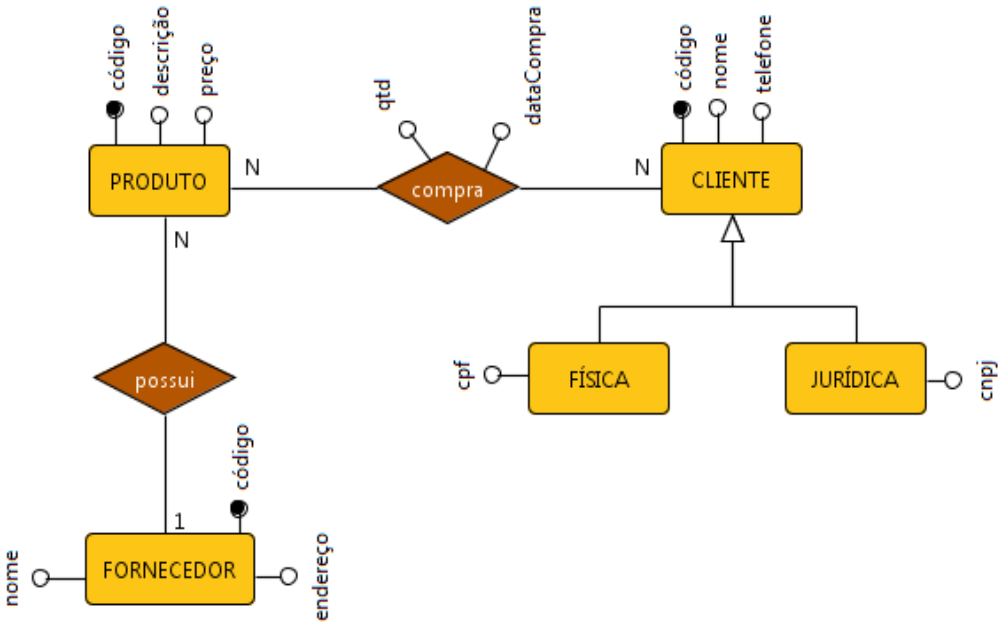


Figura 9. Diagrama entidade-relacionamento de uma loja.

As regras para realizar o mapeamento são listadas a seguir. Aplique as regras em um modelo conceitual seguindo a ordem em que as regras serão expostas.

1) Para todas as entidades do modelo conceitual deve-se criar uma tabela no modelo lógico. E os atributos de cada entidade no modelo conceitual serão as colunas da tabela criada. Os identificadores das entidades no modelo conceitual são nomeados no modelo lógico como chaves primárias e essas devem ser sublinhadas para a identificação.

   No diagrama entidade-relacionamento da Figura 9 são encontradas cinco entidades (*Produto, Fornecedor, Cliente, Física e Jurídica*). Essas entidades mapeadas para o modelo lógico ficam da seguinte forma:

   Produto (<u>código</u>, descrição, preço)

   Fornecedor (<u>código</u>, nome, endereço)

Cliente (código, nome, telefone)

Física (CPF)

Jurídica (CNPJ)

2) O próximo passo é mapear os relacionamentos do modelo conceitual para o modelo lógico. O relacionamento 1:N é mapeado para o modelo lógico incorporando a chave primária da entidade de cardinalidade 1 na entidade de cardinalidade N.
No diagrama entidade-relacionamento da Figura 9, o mapeamento do relacionamento 1:N fica da seguinte forma:

Produto (código, descrição, preço, códigoFornecedor)

códigoFornecedor referencia Fornecedor

Fornecedor (código, nome, endereço)

Cliente (código, nome, telefone)

Física (CPF)

Jurídica (CNPJ)

A chave primária da entidade Fornecedor passa a ser chave estrangeira na entidade Produto. A chave primária pode ser chamada de *Primary Key* – PK e a chave estrangeira também é conhecida pelo nome em inglês *Foreing Key* – FK.

O relacionamento 1:1 do modelo conceitual é mapeado para o modelo lógico com a incorporação da chave primária de uma entidade na outra entidade. No entanto, como a cardinalidade de ambas as entidades é 1 é possível escolher a chave primária de qualquer uma das duas entidades para incorporar na outra entidade.

O relacionamento N:N do modelo conceitual é mapeado para o modelo lógico através da criação de uma tabela e que receberão como colunas as chaves primárias das duas entidades e os

atributos de relacionamento. No diagrama entidade-relacionamento da Figura 9, o mapeamento do relacionamento N:N fica da seguinte forma:

Produto (código, descrição, preço, códigoFornecedor)
códigoFornecedor referencia Fornecedor
Fornecedor (código, nome, endereço)
Cliente (código, nome, telefone)
Física (CPF)
Jurídica (CNPJ)
Compra (códigoProduto, códigoCliente, Qtd, DataCompra)
códigoProduto referencia Produto
códigoCliente referencia Cliente

Neste caso, a tabela *Compra* foi criada e recebeu como colunas dessa tabela a chave primária das entidades *Produto* e *Cliente*. Normalmente, as chaves primárias das duas entidades passam ser a chave primária composta da tabela que foi criada. Além disso, os atributos de relacionamento do modelo conceitual se transformam em colunas na tabela criada.

3) Existem duas formas para mapear uma estrutura de Generalização/Especialização do modelo conceitual para o modelo lógico. A primeira forma é acrescentando a chave primária da entidade superclasse nas tabelas das entidades subclasses. No diagrama entidade-relacionamento da Figura 9, esse mapeamento fica da seguinte forma:

Produto (código, descrição, preço, códigoFornecedor)
códigoFornecedor referencia Fornecedor
Fornecedor (código, nome, endereço)
Cliente (código, nome, telefone)
Física (código ,CPF)

código referencia Cliente
Jurídica (código, CNPJ)
código referencia Cliente
Compra (códigoProduto, códigoCliente, Qtd, dataCompra)
códigoProduto referencia Produto
códigoCliente referencia Cliente

Nesse exemplo, a chave primária da entidade *Cliente* é acrescentada nas tabelas *Física* e *Jurídica*.

A segunda forma de realizar o mapeamento para o modelo lógico é acrescentando as colunas da entidade superclasse nas tabelas das entidades subclasses e eliminando a tabela que representa a entidade superclasse do modelo lógico. No diagrama entidade-relacionamento da Figura 9, esse mapeamento fica da seguinte forma:

Produto (código, descrição, preço, códigoFornecedor)
códigoFornecedor referencia Fornecedor
Fornecedor (código, nome, endereço)
Física (código , nome, telefone, CPF)
Jurídica (código, nome, telefone, CNPJ)
Compra (códigoProduto, códigoCliente, Qtd, dataCompra)
códigoProduto referencia Produto
códigoCliente referencia Cliente

Neste exemplo, as colunas da tabela *Cliente (código, nome, telefone)* foram acrescentadas nas tabelas *Física* e *Jurídica*. Além disso, a tabela Cliente foi eliminada.

O modelo físico é terceira etapa de um projeto de banco de dados. Esse modelo mostra com detalhes como os dados são armazenados em um banco de dados e por isso, possui o menor nível de abstração se

comparado aos demais modelos. Para definir e manipular a estrutura de banco de dados relacional é utilizado a Linguagem de Consulta Estruturada – SQL.

A SQL é uma linguagem considerada padrão para banco de dados relacional e foi desenvolvida pela IBM. A SQL pode ser dividida em dois conjuntos principais: a Linguagem de Definição de Dados – DDL e a Linguagem de Manipulação de Dados – DML. A DDL permite à criação da estrutura física que irá armazenar os dados, como as tabelas, as colunas, os índices, as chaves estrangeiras e primárias entre outras. A DML permite a inserção, a remoção, a alteração e a consulta dos registros em um banco de dados.

O Sistema Gerenciador de Banco de Dados – SGBD é software que facilita a definição, a construção, a manipulação de uma base de dados. O SGBD oferece ainda outros recursos como controle de redundância, operações de *backup* e restauração, acesso multiusuário entre outros. Dentre os SGBD disponíveis podemos citar o MySQL, PostgreSQL e o Firebird.

O MySQL será utilizado nessa e nas próximas aulas como SGBD. O MySQL Workbench é uma ferramenta gráfica para a administração de uma base de dados MySQL. Essa ferramenta será utilizada a fim de facilitar o processo de gerenciamento de um base dados MySQL através de uma interface gráfica. Antes da instalação da ferramenta do MySQL Workbench, certifique-se de que o SGBD MySQL esteja instalado.

Para instalar o MySQL Workbench, faça o *download* do instalador do *software* no endereço http://dev.mysql.com/downloads/workbench/ que seja compatível com o sistema operacional. A ferramenta Workbench exige que duas bibliotecas (*Microsoft .NET Framework 4 Client Profile* e *Redistributable Visual C ++ para o Visual Studio 2013*) estejam instaladas como pré-requisito de instalação. Caso seja necessária a instalação dessas bibliotecas, os *links* para *download* dos instaladores estão disponíveis no endereço mencionado.

Após o *download*, dê um clique duplo no instalador e o assistente de instalação será iniciado. Depois da janela de boas vindas, o usuário pode

editar o diretório de instalação. Na próxima tela do assistente, o usuário escolhe entre a instalação completa ou customizada do *software*. Recomenda-se a instalação completa do *software.* A tela inicial do *software* versão 6.3 é ilustrada na Figura 10.

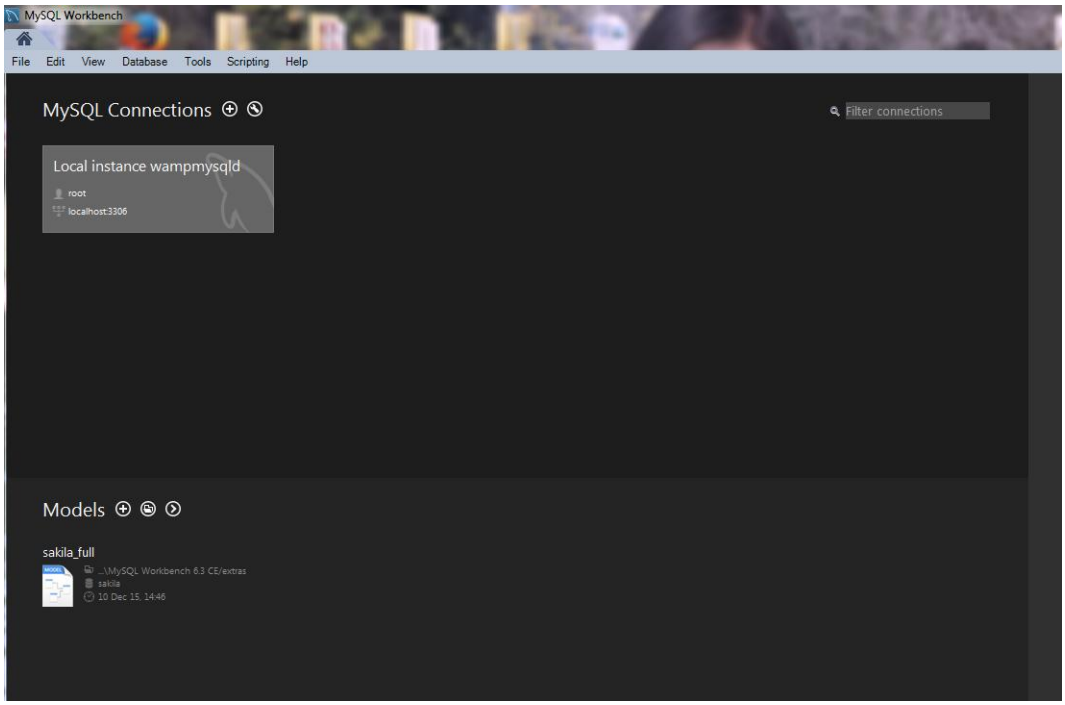


Figura 10. Tela inicial do MySQL Workbench versão 6.3.

Certifique-se que o serviço do MySQL esteja ativo. Para a execução dos comandos em SQL, clique na conexão local (retângulo cinza) e a aba ilustrada na Figura 11 será iniciada.

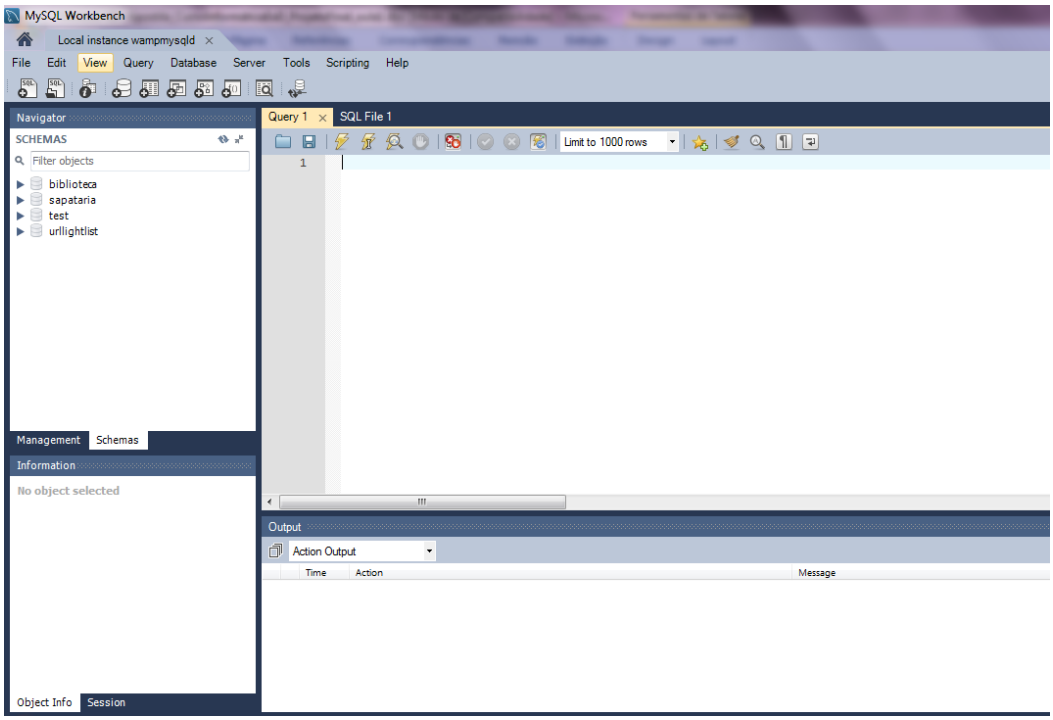


Figura 11. Aba para execução de comandos em SQL.

Em *Schemas* estão disponíveis todos os bancos de dados criados no MySQL. Para executar um comando em SQL, digite o comando na linha em azul e clique no botão com um ícone que possui o desenho de raio. Em *Output* aparece o resultado da execução do comando que pode apresentar um erro (cor vermelha) ou sucesso na execução (cor verde). A cada execução de um comando DDL é necessário atualizar o campo *Schemas,* clicando no botão de atualização para que qualquer alteração feita fique visível graficamente, ao usuário.

Após a instalação do MySQL Workbench, a estrutura física do banco de dados deve ser criada, utilizando a Linguagem de Definição de Banco de Dados – DDL. Como exemplo, será utilizado o diagrama entidade-relacionamento da Figura 9 que foi mapeado para o modelo lógico e agora será mapeado para o modelo físico.

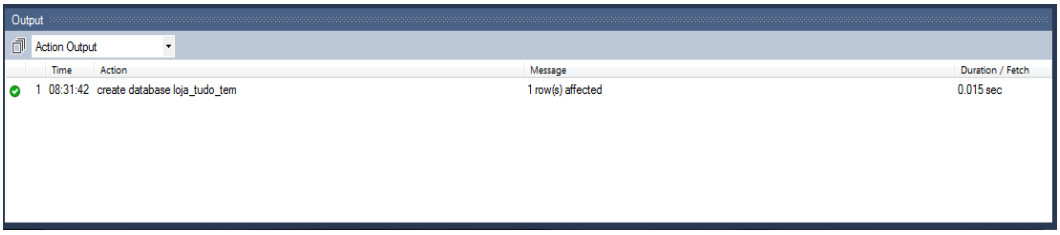
Os principais comandos DDL são: *create, alter e o drop*. O primeiro passo é criar o banco de dados que será utilizado por uma aplicação. O comando para criar um banco de dados é:

CREATE DATABASE nome_do_banco;

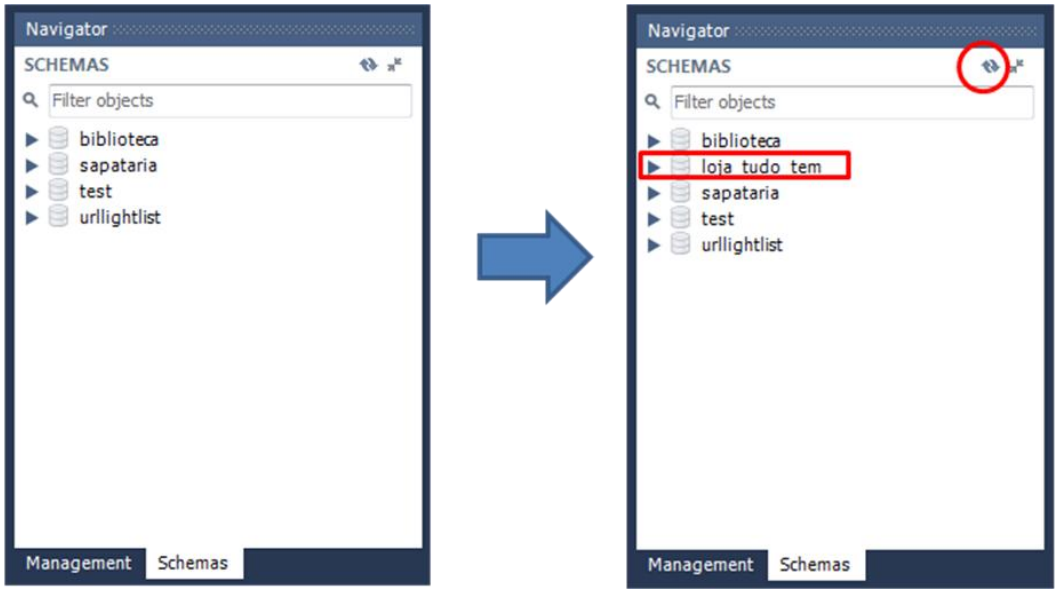
O nome escolhido para o banco de dados do exemplo será *loja_tudo_tem*. Assim, para a criação do banco de dados basta executar o comando:

Exemplo: CREATE DATABASE loja_tudo_tem;

Após a execução do comando no Workbench, uma mensagem de sucesso deve aparecer no campo *Output (*Figura 12. a) e o campo Schemas deve ser atualizado para que o banco de dados criado apareça, graficamente (Figura 12. b).



a)



b)

Figura 12. a) Campo Output com a saída de sucesso. Figura 12. b) Atualização do campo Schemas.

Para criar uma tabela de um banco de dados, selecione o banco de dados loja_tudo_tem no campo Schemas com um duplo clique e insira o comando de criação de uma tabela para a execução na aba *Query*. O comando para criar uma tabela é:

CREATE TABLE nome_do_tabela (
nome_coluna1 tipo_dados1 definições1,
nome_coluna2 tipo_dados2 definições2,
nome_colunaN tipo_dadosN definiçõesN);

Os tipos de dados permitidos pelo MySQL podem ser divididos em três grupos: numéricos, data e hora e texto. A Tabela 1 demonstra os tipos de dados numéricos permitidos no MySQL; a Tabela 2 mostra os tipos de dados data e hora e a Tabela 3 exibe os principais tipos de dados de texto.

Tabela 1. Tipos de dados numéricos no MySQL.

Tipos de dados: numéricos no MySQL	
	Descrição
TINYINT	Inteiro muito pequeno
SMALLINT	Inteiro pequeno
MEDIUMINT	Inteiro de tamanho médio
INT	Inteiro normal
BIGINT	Inteiro de tamanho grande
DECIMAL	Número decimal de ponto fixo
FLOAT	Número de ponto flutuante de precisão simples
DOUBLE	Número de ponto flutuante de precisão dupla
BIT	Um bit

Tabela 2. Tipos de dados data e hora no MySQL.

Tipos de dados: data e hora no MySQL	
Tipo	**Descrição**
DATE	Armazena uma data no formato 'YY-MM-DD'. Ex.: 1989-11-09
TIME	Armazena um valor horário no formato' hh:mm:ss'. Ex. 08:40:10
TIMESTAMP	Armazena um valor no formato 'YY-MM-DD hh:mm:ss'

Tabela 3. Tipos de dados string no MySQL.

Principais tipos de dados: string no MySQL	
Tipo	**Descrição**
CHAR	String de tamanho fixo
VARCHAR	String de tamanho variável
TEXT	String não binária

As definições mais usadas no comando *create* são:

- NOT NULL → é uma restrição que especifica que uma coluna precisa ter um valor não nulo;
- UNIQUE → é uma restrição que determina que uma coluna possua apenas valores não repetidos;
- PRIMARY KEY → a chave primária permite identificar exclusivamente uma tabela;
- FOREIGN KEY → a chave estrangeira em uma coluna da tabela precisa combinar com uma chave primária existente na tabela referenciada;
- AUTO_INCREMENT→ é usado para gerar uma identificação única para um novo registro.

Para criar as tabelas mapeadas na Figura 9 será necessária a execução dos seguintes comandos DDL no Workbench:

- Comando para a criação da tabela Fornecedor:

```
create table Fornecedor(
codigo int auto_increment primary key,
nome varchar(50) not null,
endereco varchar(20) not null
)
```

- Comando para a criação da tabela Produto:

```
create table Produto(
codigo int auto_increment primary key,
codigoFornecedor int,
descricao varchar(20) unique not null,
preco float not null,
```

```
foreign key (codigoFornecedor) references Fornecedor(codigo)
)
```

- Comando para a criação da tabela Cliente:

```
create table Cliente(
codigo int auto_increment primary key,
nome varchar(30) not null,
telefone varchar(10)
)
```

- Comando para a criação da tabela Física:

```
create table Fisica(
codigo int,
cpf varchar(11) unique not null,
foreign key (codigo) references Cliente(codigo)
)
```

- Comando para a criação da tabela Jurídica:

```
create table Juridica(
codigo int,
cnpj varchar(14) unique not null,
foreign key (codigo) references Cliente(codigo)
)
```

- Comando para a criação da tabela Compra:

```
create table Compra(
codigoProduto int,
codigoCliente int,
qtd int not null,
dataCompra date not null,
foreign key (codigoProduto) references Produto(codigo),
foreign key (codigoCliente) references Cliente(codigo)
)
```

A Figura 13 mostra o campo Schemas após a criação das tabelas do banco de dados *loja_tudo_tem* e a atualização do campo Schemas.

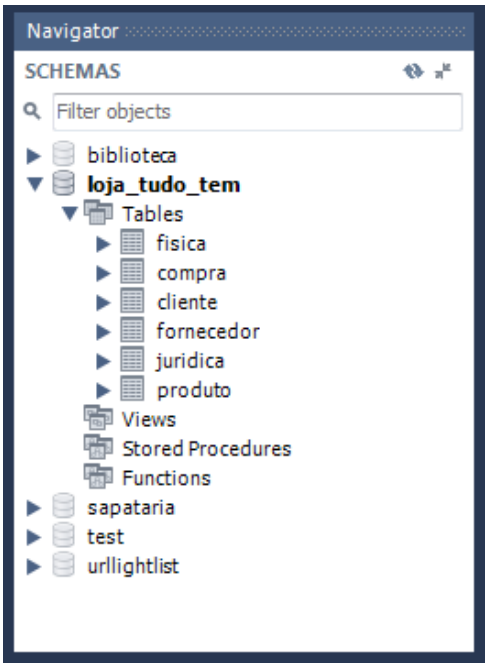


Figura 13. Criação das tabelas no Workbench.

O comando para alterar uma tabela adicionando mais uma coluna é: **ALTER TABLE nome_tabela ADD nome_coluna tipo_dados definições**. Para adicionar o campo CNPJ na tabela Fornecedor, por exemplo, o comando em SQL ficará da seguinte forma:

Exemplo: ALTER TABLE FORNECEDOR ADD CNPJ VARCHAR (14) UNIQUE NOT NULL;

Para excluir uma tabela, selecione o banco de dados com um clique duplo no campo Schemas e execute o seguinte comando: **DROP TABLE nome_tabela;**

Os principais comandos em DML são: o *insert*, o *delete*, o *update* e o *select*. Para inserir um registro em uma tabela, selecione o banco de dados com um clique duplo no campo Schemas e execute o seguinte comando: **INSERT INTO nome_tabela (campo1, campo2, campoN) VALUES (valor1, valor2, valorN);**

Por exemplo, para inserir o registro de um cliente na tabela cliente, excute o comando: **INSER INTO cliente (nome, telefone) VALUES ('Maria Dolores da Rosa', '2345-9878');**

Para atualizar um registro em uma tabela, execute o comando: **UPDATE nome_tabela SET campo1= novoValor1, campo2= novoValor2, campoN= novoValorN WHERE condição;.** Por exemplo, para alterar o registro da cliente Maria Dolores da Rosa que possui como código o valor um, excute o comando: **UPDATE cliente SET telefone='2345-7869' WHERE codigo=1;.**

Para excluir o registro de um cliente, execute o seguinte comando: **DELETE FROM nome_tabela WHERE comando;** Por exemplo, para excluir o registro da cliente Maria Dolores da Rosa, excute o comando: **DELETE FROM cliente WHERE código=1;**

Para recuperar os registros de uma tabela, o comando *select* é utilizado. A sintaxe mais simples desse comando é: **SELECT campo1, campo2, campoN FROM nome_tabela WHERE condição;**

Operadores aritméticos e lógicos podem ser utilizados na cláusula *where.* Para que todos os campos de uma tabela sejam recuperados utilize o caractere asterisco no lugar das especificações do campo e para recuperar registros sem duplicação utilize a cláusula *distinct*.

Para recuperar os campos *nome* e *telefone* de todos os clientes em ordem alfabética, execute o seguinte comando: **SELECT nome, telefone FROM cliente ORDER BY nome ASC;.** A Figura 14 mostra o resultado da execução desse comando no Workbench.

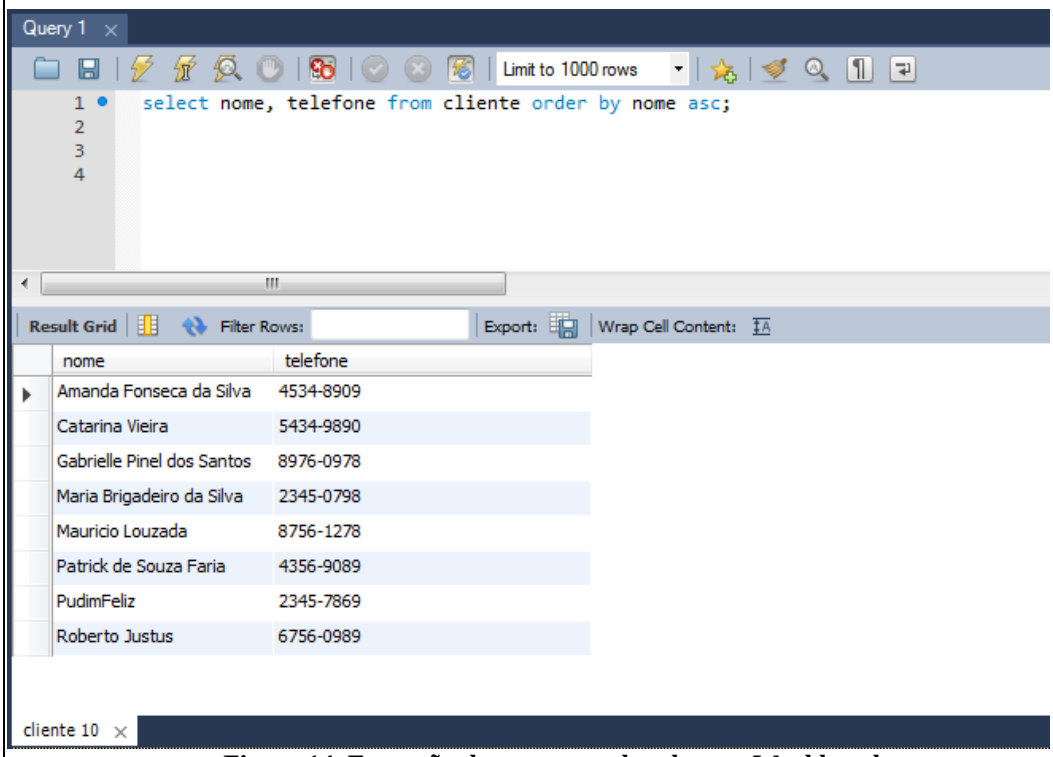


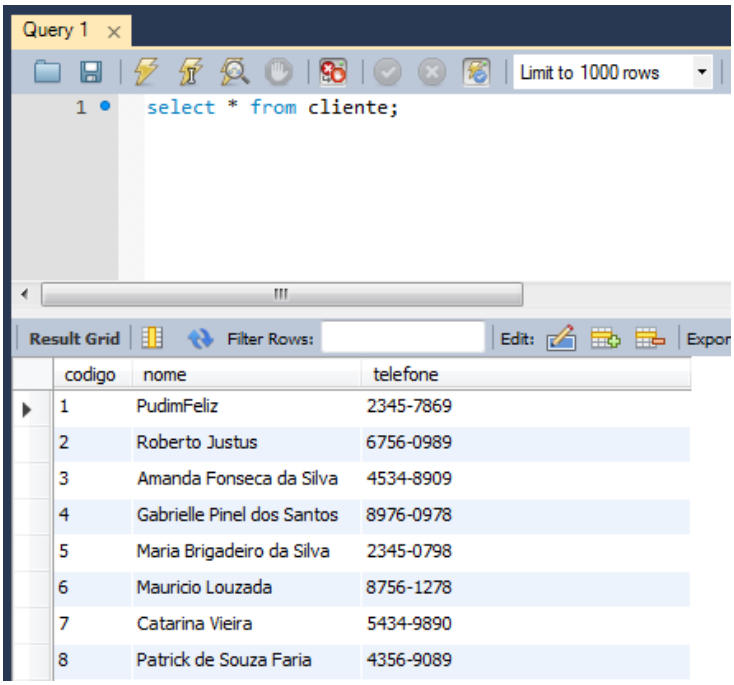
Figura 14. Execução de um comando select no Workbench.

Algumas funções são permitidas no SQL. Geralmente, essas funções são utilizadas em conjunto com a cláusula *Group By* que possibilita agrupar o resultado por uma ou mais colunas. Já a cláusula *Having* especifica uma condição para o resultado de um comando Group By. As funções disponíveis no SQL são:

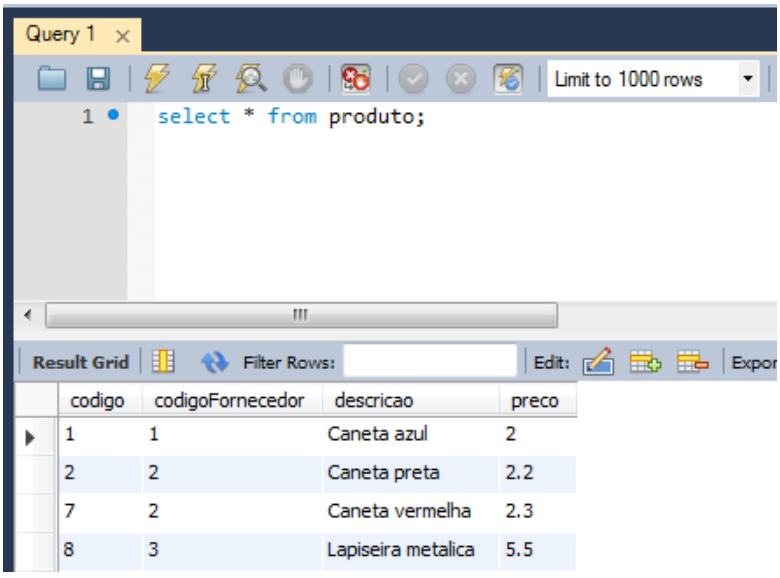
- COUNT → retorna o número de linhas afetadas;
- MAX → retorna o maior valor de uma coluna;
- MIN → retorna o menor valor de uma coluna;
- AVG → retorna a média de valores;
- SUM → retorna a soma de valores.

Por exemplo, o comando para recuperar a descrição dos produtos e a respectiva quantidade de produtos comprados, sendo que a quantidade comprada deve ser superior a 155 por produto fica da seguinte forma: **select p.descricao, sum(qtd) as QTD from compra c, produto p, cliente cli where c.codigoProduto=p.codigo and c.codigoCliente=cli.codigo group by p.descricao having QTD > 155;**

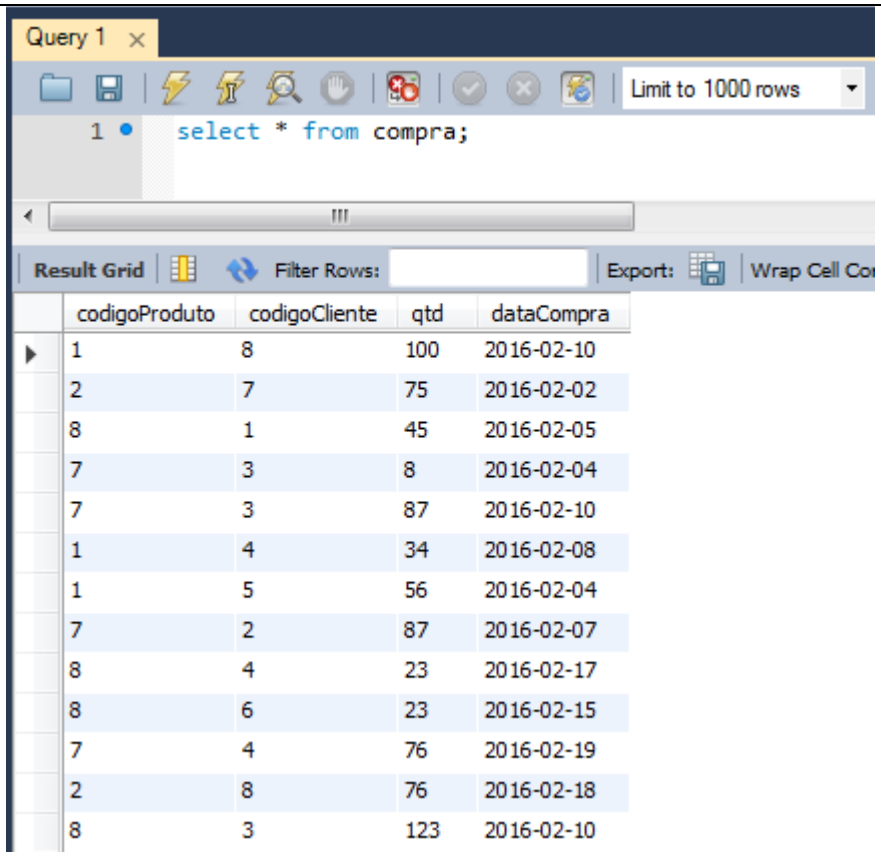
As Figuras 15. a), 15. b) 15. c) exibem, respectivamente, os registros das tabelas *Cliente*, *Produto* e *Compra.* E a Figura 15. d) mostra o resultado da execução do comando *select* do exemplo.



a)

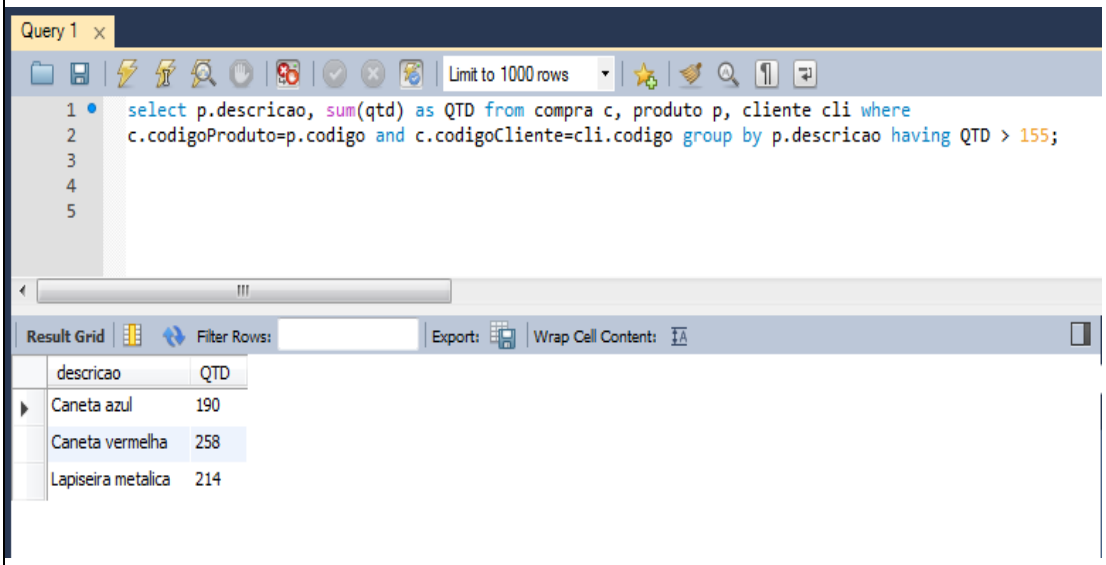


b)



codigoProduto	codigoCliente	qtd	dataCompra
1	8	100	2016-02-10
2	7	75	2016-02-02
8	1	45	2016-02-05
7	3	8	2016-02-04
7	3	87	2016-02-10
1	4	34	2016-02-08
1	5	56	2016-02-04
7	2	87	2016-02-07
8	4	23	2016-02-17
8	6	23	2016-02-15
7	4	76	2016-02-19
2	8	76	2016-02-18
8	3	123	2016-02-10

c)



descricao	QTD
Caneta azul	190
Caneta vermelha	258
Lapiseira metalica	214

d)

Figura 15. a) Registros da tabela cliente. Figura 15. b) Registros da tabela produto. Figura 15. c) Registros da tabela compra. Figura 15. d) Resultado da execução do comando select do exemplo no Workbench.

Para recuperar a descrição dos produtos que começam com a letra

"C" com as suas respectivas quantidades compradas entre as datas de 01/02/2016 a 15/02/2016 execute o seguinte comando: **select p.descricao, SUM(qtd) as QTD from compra c, produto p, cliente cli where c.codigoProduto=p.codigo and c.codigoCliente=cli.codigo and p.descricao like 'C%' and c.dataCompra between '2016/02/01' and '2016/02/15' group by p.descricao;.** A Figura 16 ilustra a execução do comando *select* mencionado.

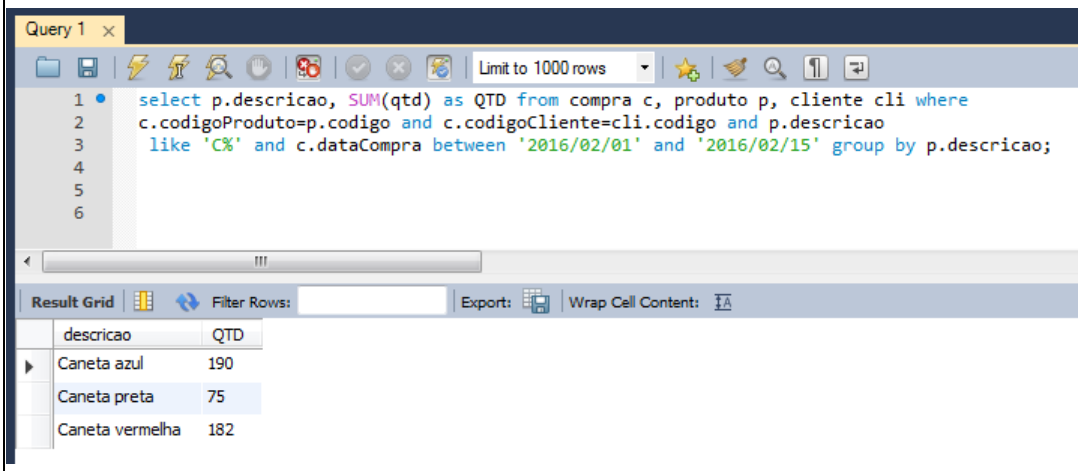


Figura 16. Resultado do comando select no Workbench.

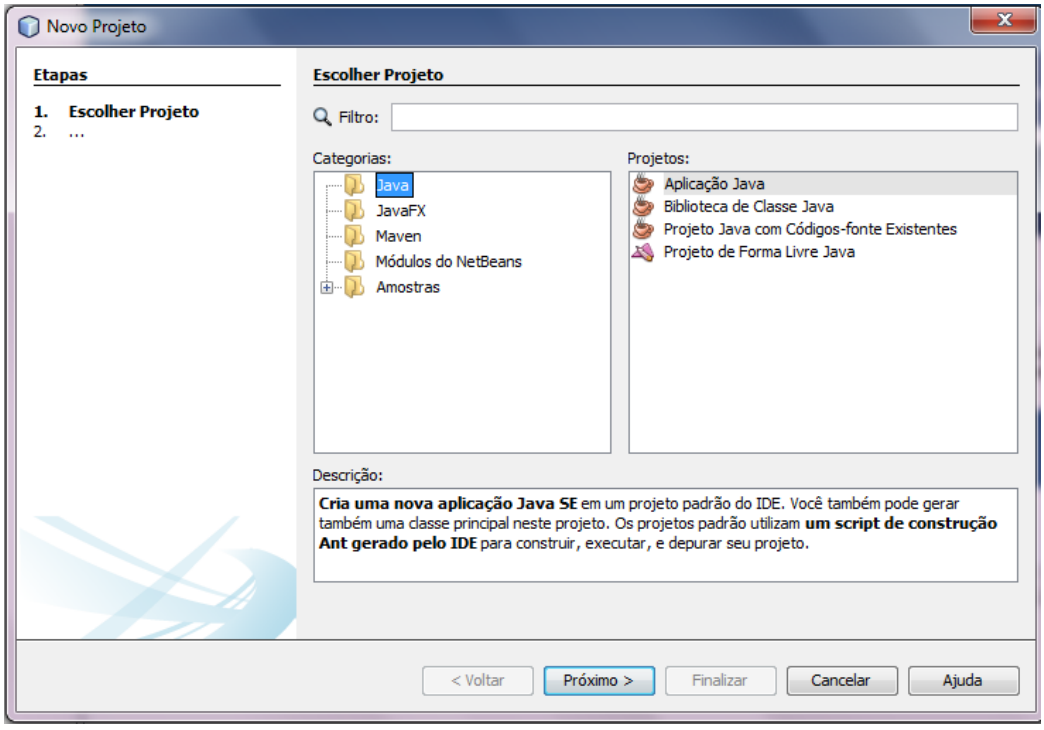
Aula 3

Criação de interfaces gráficas

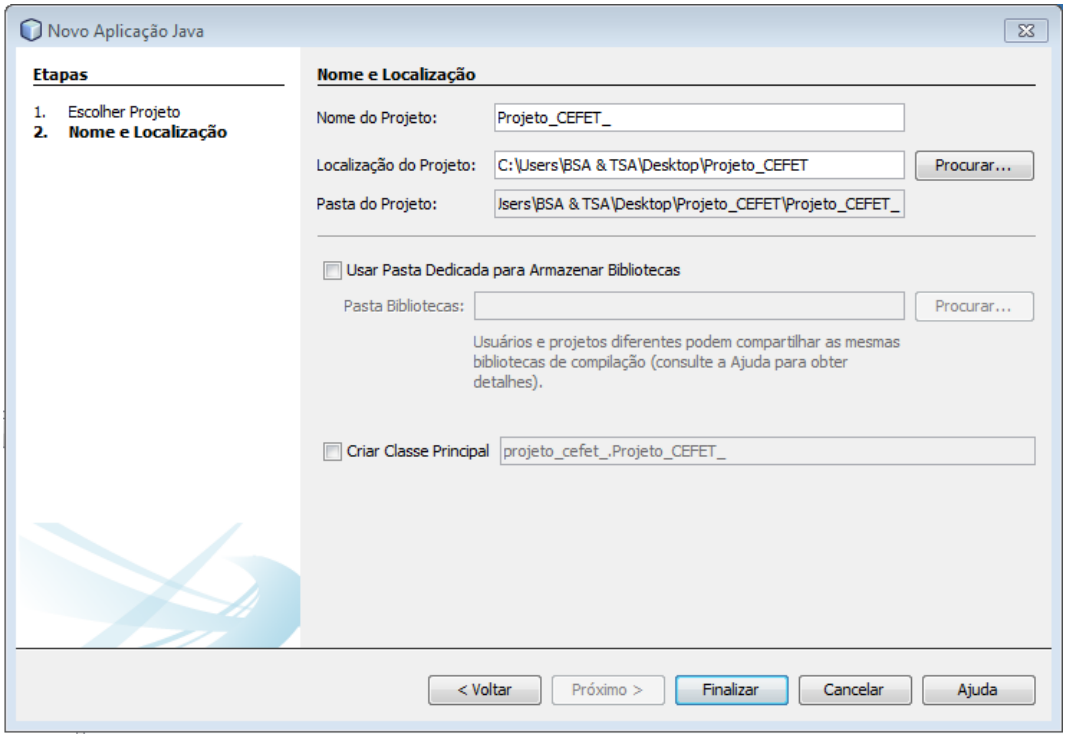
3. Criando uma interface gráfica no NetBeans.

Nesta aula, será demonstrado passo a passo como criar uma interface gráfica utilizando o ambiente de desenvolvimento integrado (IDE) do NetBeans. Essa IDE permite o desenvolvimento rápido de aplicações em Java SE já que as bibliotecas gráficas AWT e Swing vem pré-instaladas (NetBeans, 2016). Os componentes das bibliotecas gráficas no NetBeans podem ser manuseados com o auxílio de uma paleta gráfica, simplesmente, arrastando os componentes necessários para uma determinada tela.

A versão do NetBeans IDE que será usada é a 8.1 e está disponível para o *download* gratuitamente, no seguinte endereço: https://netbeans.org/downloads/. Para criar um novo projeto em Java SE no NetBeans, clique em na aba *Arquivo* e escolha a opção *Novo Projeto.* Na janela que foi aberta selecione em *Categorias* a opção *Java* e em *Projetos* a opção a *Aplicação Java* (Figura 17. a). Na próxima tela, digite um nome para o projeto, escolha um diretório para armazenar a aplicação que será desenvolvida e desmarque a opção *Criar Classe Principal* (Figura 17. b).



a)



b)

Figura 17. a) Primeira tela para a criação de um projeto em Java SE no NetBeans. Figura 17. b) Segunda tela para a criação de um projeto em Java SE no NetBeans.

Nesta aula será mostrada a criação das telas ilustradas na Figura 18. a) e 18. b) pertencentes ao sistema de gerenciamento da clínica veterinária *Bom Pra Cachorro*. A tela principal (Figura 18. a) é a tela de inicialização da aplicação e possui um menu com as funcionalidades disponíveis no sistema (*Cadastro, Relatórios, Consulta*). No canto inferior direito, a tela principal exibe a data atual.

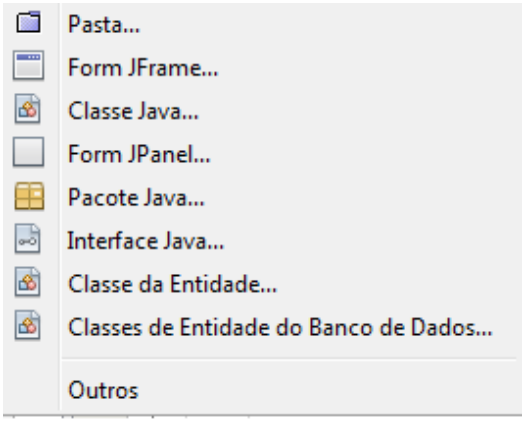
A tela da Figura 18. b) é um formulário para o cadastro de um médico veterinário, onde os campos *nome, CRVM* e *formação* devem ser informados pelo usuário do sistema. Se o botão *Salvar* for acionado, a aplicação deve armazenar os dados informados pelo usuário no banco de dados; se o botão *Cancelar* for acionado, a tela de cadastro é fechada e a tela principal será exibida e se o botão *Limpar* for pressionado, os campos preenchidos serão apagados.

a)

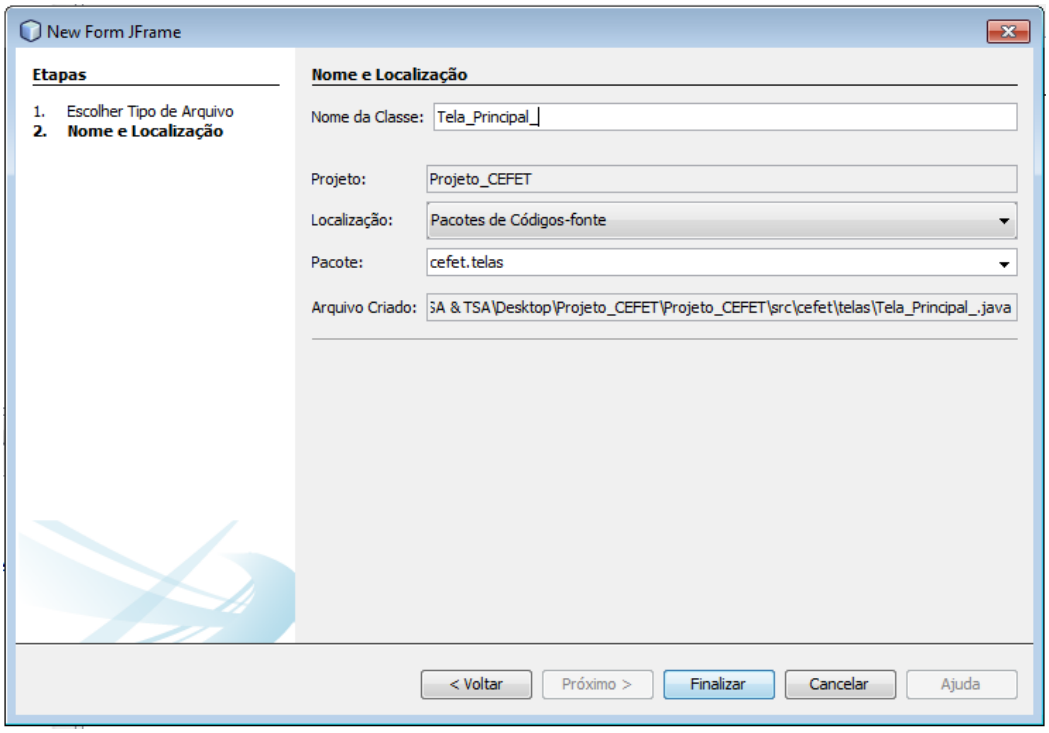
b)

Figura 18. a) Tela principal do sistema da clínica veterinária Bom Pra Cachorro. Figura 18. b) Tela de cadastro de um médico veterinário.

Primeiramente, para criar uma tela no NetBeans clique com o botão direito do *mouse* no nome do projeto e escolha a opção *Novo*. A Figura 19. a) ilustra as opções disponíveis, escolha a opção *Form JFrame.* Na tela seguinte (Figura 19. b), basta nomear a classe a ser criada (*Tela_Principal*) e escolher ou criar um novo pacote (*cefet.telas*). O *JFrame* é um container que representa uma janela onde é possível montar uma aplicação anexando outros elementos.



a)



b)

Figura 19. a) Opções disponíveis em Novo. Figura 19. b) Tela para a criação de um Form JFrame.

A Figura 20 exibe a IDE do NetBeans, após a criação de um JFrame. Na aba Código-Fonte, o programador acessa o código referente à classe criada (atributos e métodos). Na aba projeto, o programador pode acessar o editor gráfico do NetBeans. Os componentes das bibliotecas gráficas (Swing e AWT) estão disponíveis na *Paleta de Componentes* e para inserir um componente, basta arrastá-lo e posicioná-lo na janela. No ícone *Visualizar Design* (circulado) é possível visualizar a aparência e a organização dos componentes da janela que está sendo criada.

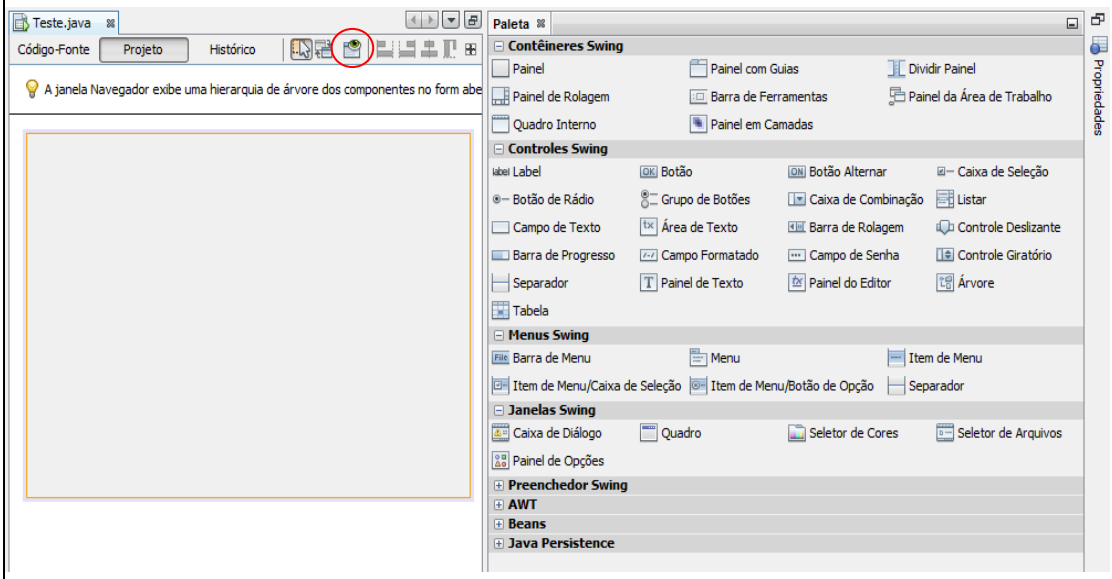


Figura 20. Editor gráfico do NetBeans.

Os componentes que serão utilizados na criação das janelas ilustradas nas Figuras 18. a) e 18. b) são:

- JFrame → (já criado);
- Label → componente utilizado para inserir um rótulo ou ainda uma imagem em uma janela. Este componente está localizado na aba *Controles Swing* da *Paleta de Componentes;*
- Botão → componente usado para disparar um evento em uma janela. Este componente está localizado na aba *Controles Swing* da *Paleta de Componentes;*
- Barra de Menu → componente que possibilita a criação de uma

árvore de menu com itens e subitens (Figura 21). Em cada item e subitem é possível adicionar eventos. Este componente está localizado na aba *Menus Swing* da *Paleta de Componentes;*

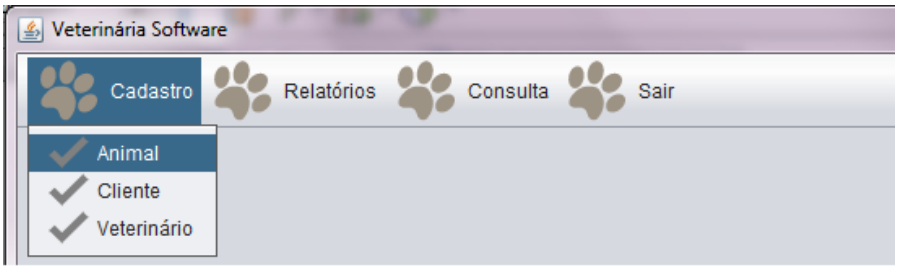


Figura 21. Barra de Menu.

- Campo de Texto → componente que permite ao usuário inserir textos. Este componente está localizado na aba *Controles Swing* da *Paleta de Componentes;*
- Caixa de Combinação → componente que permite ao usuário selecionar um valor dentre as opções disponíveis. Este componente está localizado na aba *Controles Swing* da *Paleta de Componentes.*

Depois que um componente é inserido na IDE do NetBeans, as configurações e as propriedades desse componente podem ser modificadas através de uma aba chamada de *Propriedades* (Figura 22). Essa aba pode ser acessada clicando com o botão direito do *mouse* em cima do componente e escolhendo a opção propriedades.

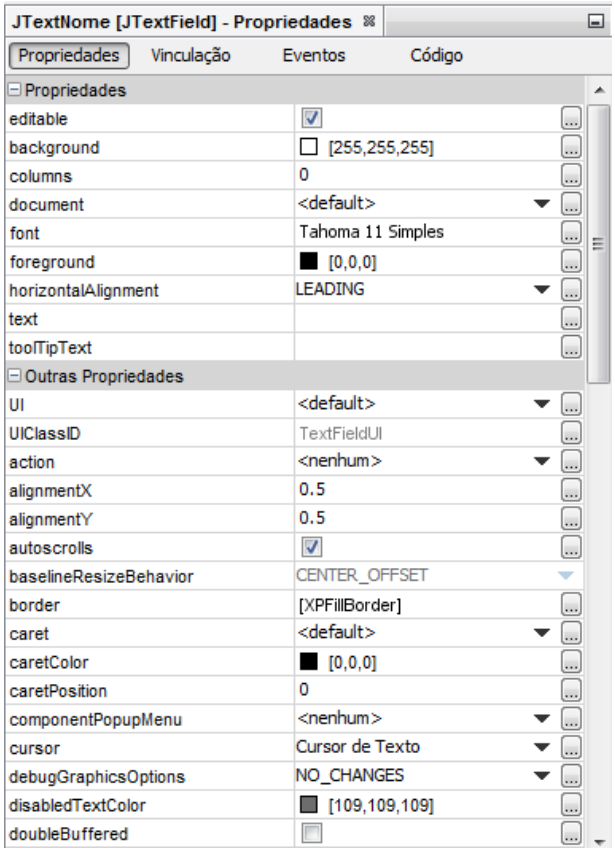


Figura 22. Tabela de edição das propriedades de um componente no NetBeans.

Nessa aba é possível alterar todas as características de um componente. Por sua vez, cada característica altera de alguma forma o modo de exibição do componente. As seguintes propriedades de componentes foram modificadas na tela principal e de cadastro:

- *defaultCloseOperation:* a classe *JFrame* suporta quatro opções quando o usuário da aplicação fecha uma janela. A *HIDE* é a opção *default* a qual oculta à janela; a opção *DISPOSE* descarta a janela devolvendo os recursos; a opção *DO_NOTHING* possibilita ao programador determinar o que ocorrerá quando a janela for fechada e a opção *EXIT_ON_CLOSE* finaliza a aplicação se a janela for fechada;
- *title:* possibilita a inserção de um título em um JFrame que aparecer no canto superior esquerdo. Nas janelas do exemplo, os títulos são: *Veterinária Software* e *Cadastro Veterinário;*
- *font:* essa propriedade possibilita a alteração do tipo e do

tamanho da fonte;

- *resizable*: permite que o JFrame seja redimensionável ou não. Caso esta propriedade seja desabilitada, o botão de maximizar da janela aparecerá inativo;
- *Tamanho do Designer*: Para definir a largura e a altura de um JFrame, selecione a opção *Código* da janela de propriedade e altere os valores do campo *Tamanho de Designer.* Na tela principal, a largura e a altura foram alteradas para respectivamente, 800 e 600.
- *Gerar Centralizado:* possibilita que um JFrame apareça de forma centralizada no momento de execução do aplicativo. Para configurar essa propriedade, selecione a opção *Código* da janela de propriedade;
- *Nome da Variável:* Para manter a organização e uma melhor estruturação do código, nomeie todos os componentes inseridos. Para configurar essa propriedade, selecione a opção *Código* da janela de propriedade. Utilize um padrão para nomear os componentes como, por exemplo, para nomear os componentes Label pode-se utilizar do seguinte padrão: JLabelNomeDaVariavel. Ex.: JLabelNome
- *toolTipText:* essa propriedade permite a exibição de uma informação adicional sobre o componente quando o *mouse* é posicionado sobre esse, no momento da execução (Figura.

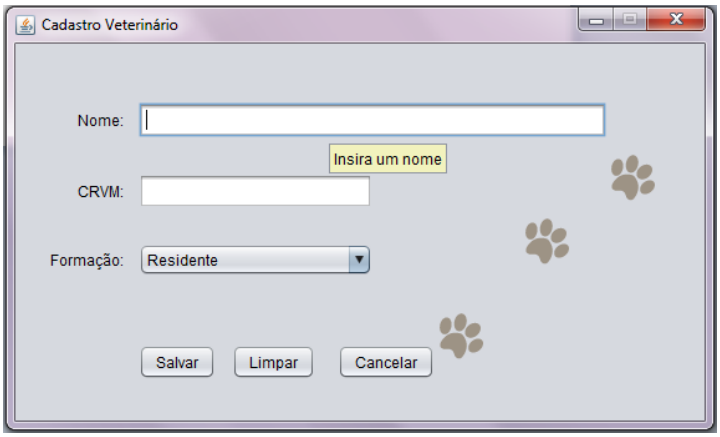


Figura 23. Propriedade toolTipText.

- *Icon* → possibilita a inserção de uma imagem. As imagens foram inseridas nos componentes Label e na Barra de Menu (menu e itens). A janela para inserção dessa propriedade está ilustrada na Figura 24.

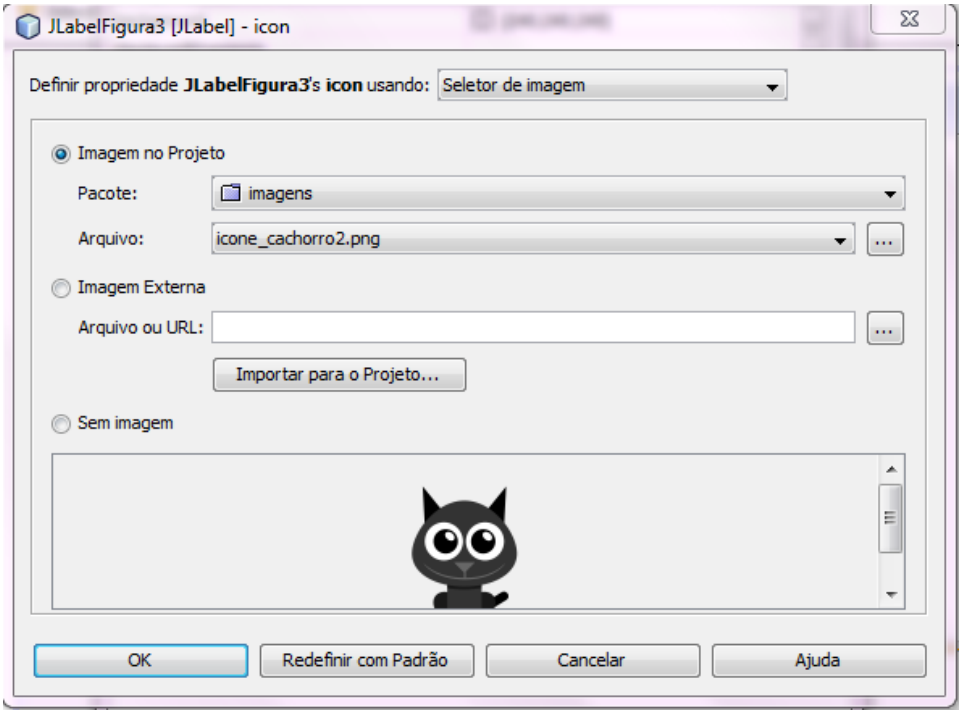


Figura 24. Janela para a configuração da propriedade Icon.

- *Text*: essa propriedade define o texto que é exibido no componente. Essa propriedade foi usada no componente Button e Label. Ex.: Salvar
- *Model*: essa propriedade possibilita a definição dos itens que estarão disponíveis para a seleção no componente de Caixa de Seleção. Para definir os itens, basta escrevê-los um em cada linha.

Um evento permite que uma ação programada seja executada após o disparo feito pelo usuário ou por um gatilho interno. Para verificar os eventos disponíveis para cada componente, clique com o botão direito do *mouse* em cima do componente, selecione a opção *Propriedades* e depois

a aba *Eventos.* Por exemplo, todos os eventos disponíveis do componente botão estão ilustrados na Figura 25.

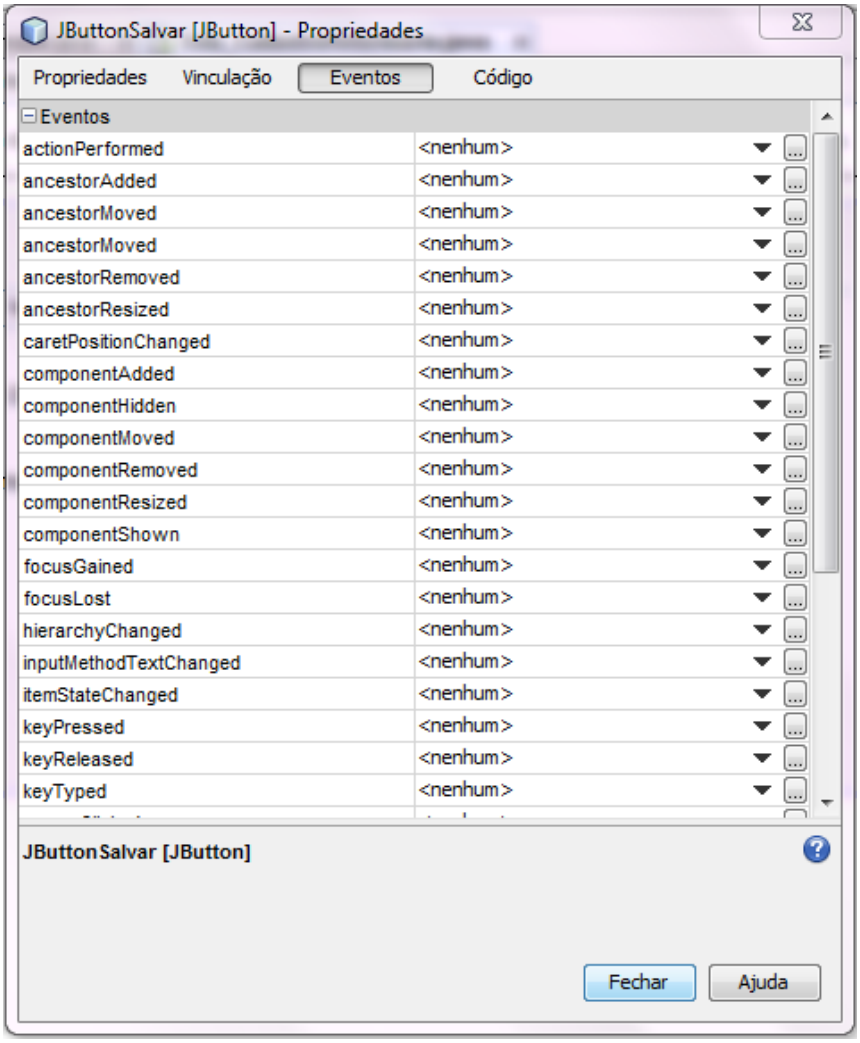


Figura 25. Eventos disponíveis para o componente botão.

Por exemplo, para adicionar um evento que dispara quando o usuário clicar no botão *Salvar* da tela de *Cadastro*, selecione o evento *actionPerformed* na aba de *Evento*, adicione um nome (*Salvar_CadastroVeterinario*) e clique no botão *OK.* Automaticamente, um método com o nome atribuído será criado referente ao evento escolhido. Esse código pode ser visualizado na aba *Código-Fonte* (Figura 26).

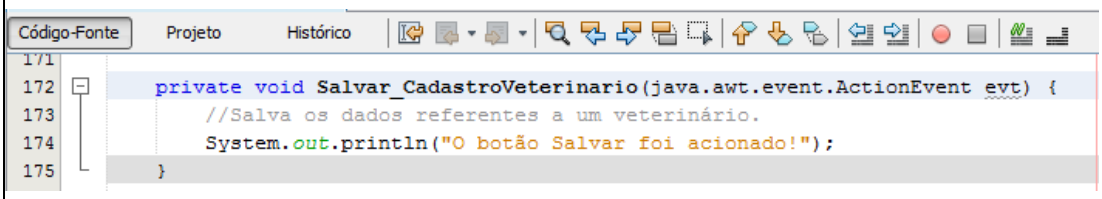


Figura 26. Código gerado pelo evento *actionPerformed*.

Note que um comentário e uma instrução foram adicionados ao método Salvar. A instrução *System.out.println* gera uma saída texto quando o método for executado. Para executar o aplicativo, clique no botão *Executar Projeto* que possui o símbolo de um *play*. Se o botão *Salvar da tela de Cadastro* for acionado, a saída do comando é exibida Figura 27.

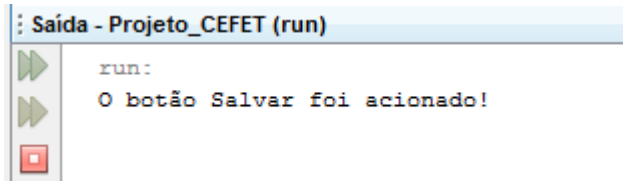


Figura 27. Saída do comando *Salvar*.

Os eventos adicionados à tela *Principal* serão exibidos nas Figuras 28, 29 e 30. Em cada código é exibido um comentário explicativo sobre o método.

```
private void carregarFrame(java.awt.event.WindowEvent evt) {
    //Exibe a data atual ao carregar a janela principal
    Date dataAtual= new Date();
    SimpleDateFormat formato = new SimpleDateFormat("dd/MM/YY");
    JLabelData.setText(formato.format(dataAtual));
}
```

Figura 28. Evento *windowOpened* adicionado ao JFrame.

```
private void abrir_CadastroVeterinario(java.awt.event.ActionEvent evt) {
  //Exibe a tela Veterinário e fecha a tela principal
  //O objeto da tela principal é passado por paramêtro no construtor da tela Veterinário
   Tela_CadastroVeterinário tela_cadastro_veterinario = new Tela_CadastroVeterinário(this);
   tela_cadastro_veterinario.setVisible(true);
   this.setVisible(false);
}
```

Figura 29. Evento *actionPerformed* adicionado ao subitem Veterinário da barra de menu.

```
  private void Sair_Aplicacao(java.awt.event.ActionEvent evt) {
      //Sair da aplicação
      System.exit(0);
  }
```

Figura 30. Evento *actionPerformed* adicionado ao item Sair da barra de menu.

Os eventos adicionados à tela *Cadastro de Veterinário* serão exibidos nas Figuras 31, 32 e 33. Em cada código é exibido um comentário explicativo sobre o método. O código do botão *Salvar* é exibido na Figura 26.

```
//Atributo criado com o tipo da tela Principal
private Tela_Principal telaprincipal;
//Construtor com parâmetro da tela Cadastro de Veterinário
//Atributo da tela principal recebe o parâmetro do construtor
  public Tela_CadastroVeterinário(Tela_Principal telaPrincipal) {
      this.telaprincipal=telaPrincipal;
      initComponents();
  }

   public Tela_CadastroVeterinário() {
      initComponents();
  }
```

Figura 31. Construtor com parâmetro da tela de Cadastro de Veterinário.

```
private void limpar_TelaVeterinario(java.awt.event.ActionEvent evt)
//Apaga os campos preenchidos na Tela de Cadastro de um Veterinário
    JTextNome.setText("");
    JTextCRVM.setText("");
    JComboFormação.setSelectedItem(null);
}
```

Figura 32. Evento *actionPerformed* adicionado ao botão Limpar.

```
private void cancelar_CadastroVeterinario(java.awt.event.ActionEvent evt) {
  //Cancela o casdastro.
    System.out.println("A inserção dos dados foi cancelada!");
    this.setVisible(false);
    this.telaprincipal.setVisible(true);
}
```

Figura 33. Evento *actionPerformed* adicionado ao botão *Cancelar*.

Entendendo o padrão Model-View-Controller (MVC)

4. Entendo o padrão *Model-View-Controller* (MVC).

Nesta aula, será abordado o padrão *Model-View-Controller* (MVC) que é um padrão de desenvolvimento de *software* que separa a implementação do *software* em camadas. Será apresentada a principal ideia e o objetivo que envolve esse padrão de desenvolvimento. Além disso, os passos iniciais para utilizar o modelo MVC serão demonstrados e a camada *Data Access Object* (*DAO)* que permite o acesso ao banco de dados será implementada.

Segundo Gamma (2000), o modelo MVC é composto por três classes: Modelo, Visão e Controlador. O Modelo é o objeto de aplicação, a Visão são as interfaces gráficas que interagem com o usuário e o Controlador é o intermediário entre as camadas de Visão e a Modelo. Sendo assim, o Modelo representa a lógica da aplicação com as regras de negócio; a Visão recebe e exibe as informações aos usuários através das interfaces gráficas e o Controlador controla o fluxo de informação da Visão para o Modelo.

O MVC possibilita a organização do código em camadas de forma que cada camada possui uma funcionalidade específica. Antes desse padrão, um único módulo agrupava todas essas funcionalidades. Esse módulo único possui muitas linhas de código dificultando o entendimento, a manutenção e a reutilização do código. O padrão MVC proporcionou a independência entre a lógica de negócio (Modelo) e a interface com o usuário (Visão).

Para exemplificar a implementação do modelo MVC será utilizado o *Create*, *Read*, *Update* e *Delete* (CRUD) de veterinário. Para cadastrar os dados de um profissional veterinário, o usuário terá que informar o nome, o CRVM e a formação. A interface gráfica da tela de cadastro de veterinário desse exemplo foi construída na aula 3.

O primeiro passo para desenvolver um *software* segundo o padrão MVC é criar os pacotes correspondentes a cada camada do modelo MVC. Esses pacotes agrupam as classes que desempenham as mesmas funcionalidades. Para isso, crie no NetBeans quatro pacotes: *view*,

controller, model e dao, conforme ilustrado na Figura 34.

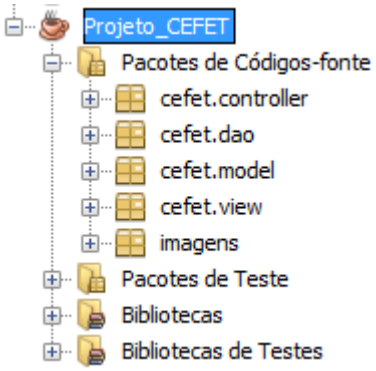


Figura 34. Pacotes criados no NetBeans.

No pacote *model,* crie a classe *Veterinario* com os atributos *id*, *nome*, *CRVM, formação* e os métodos construtores, *getters* e *setters.* O código da classe *Veterinario* é ilustrado na Figura 35.

```
public class Veterinario {
    private int id_veterionario;
    private String nome;
    private String CRVM;
    private String formacao;
    //Construtores
    public Veterinario(){  }
    public Veterinario(String nome, String CRVM, String formacao){
      this.nome=nome;
      this.CRVM=CRVM;
      this.formacao=formacao;
    }
    public int getId_veterionario() {return id_veterionario;}
    public void setId_veterionario(int id_veterionario){this.id_veterionario = id_veterionario; }
    public String getNome() {return nome;}
    public String getCRVM() {return CRVM;}
    public String getFormacao() {return formacao;}
    public void setNome(String nome){this.nome = nome;}
    public void setCRVM(String CRVM) {this.CRVM = CRVM;}
    public void setFormacao(String formacao) {this.formacao = formacao;}
}
```

Figura 35. Código da classe *Veterinario*.

As classes pertencentes ao pacote *view* que serão utilizadas nesse exemplo foram criadas na aula três. As interfaces gráficas chamadas de *Tela_Principal* e *Tela_CadastroVeterinario* pertencem ao pacote *view* (Figura 36).

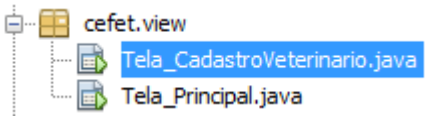


Figura 36. Classes do pacote *view*.

As classes pertencentes ao pacote *dao* permitem o acesso ao banco de dados e separam a camada de Modelo do MVC da camada de dados. A classe *ConexaoBancoDeDados* desse pacote estabelece uma conexão com o banco de dados através do método *getConnection()*. A Figura 37 exibe o código para a conexão com o banco de dados.

```
public class ConexaoBancoDeDados {

    @SuppressWarnings("CallToPrintStackTrace")
    public Connection getConnection(){

                Connection conn = null;

                try{
                    Class.forName("com.mysql.jdbc.Driver");
                }
                catch (Exception e) {
                        e.printStackTrace();
                }
                try {

                conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/clinicaveterinaria","root",null);

                }catch (SQLException e) {
                e.printStackTrace();
                System.out.println(e);
        }
                return conn;
        }

}
```

Figura 37. Código da classe *ConexaoBancoDeDados*.

Note que no método *getConnection()* foi definido o banco de dados (*clinicaveterinaria*) e o servidor (*localhost).* Na base de dados *clinicaveterinaria* foi criada uma tabela chamada de *veterinario* com as seguintes colunas *id*, *nome*, *crvm* e formação. Na classe *ConexaoBancoDeDados* é necessário realizar as importações de classes ilustradas na Figura 38.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
```

Figura 38. Importações realizadas na classe *ConexaoBancoDeDados.*

Após a criação da classe *ConexaoBancoDeDados* é necessário adicionar o Driver de conexão do MySQL. Para isso, clique com o botão direito do *mouse* no nome do projeto e selecione a opção *Propriedades.* Na janela aberta, escolha a categoria *Bibliotecas* e clique no botão *Adicionar Bibliotecas.* Agora basta escolher a opção *Driver JDBC do MySQL*

Para cada classe do Modelo será criada uma classe correspondente no pacote *dao*. Essa classe possui os métodos que fazem acesso ao banco de dados possibilitando a inserção, a exclusão, a alteração e a seleção de dados de um determinado objeto. Neste caso, a classe *VeterinarioDAO* será criada para implementar o *Create, Read, Update* e *Delete* (CRUD) de veterinário.

Na classe *VeterinarioDAO* crie um método chamado de *cadastrarVeterinario* sem tipo de retorno e que receba como parâmetro um objeto do tipo *Veterinario.* A Figura 39 exibe o código do método *cadastrarVeterinario*.

```
public void cadastrarVeterinario(Veterinario veterinario) throws ExceptionDAO{

        String sql = "insert into veterinario (nome,crvm,formacao) values (?,?,?)";
        PreparedStatement stmt = null;
        Connection connection = null;
        try {
                connection= new ConexaoBancoDeDados().getConnection();
                stmt = connection.prepareStatement(sql);
                stmt.setString(1, veterinario.getNome());
                stmt.setString(2, veterinario.getCRVM());
                stmt.setString(3, veterinario.getFormacao());
                stmt.execute();

        } catch (SQLException e) {
                e.printStackTrace();
                throw new ExceptionDAO("Erro ao cadastrar veterinário:"+ e);

        } finally {
                try {if (stmt != null) {stmt.close();}
                } catch (SQLException e) { e.printStackTrace();}
                try {if (connection != null) {connection.close();}
                } catch (SQLException e) {e.printStackTrace();}
        }

}
```

Figura 39. Método *cadastrarVeterinario* da classe *VeterinárioDAO*.

A classe *ConexaoBancoDeDados* do pacote *dao* é instanciada para que através do método *getConnection()* uma conexão com o banco de dados seja estabelecida. E o comando em *Structured Query Language* (SQL) (*PreparedStatement*) é preparado com as informações do objeto da camada de Modelo (*Veterinario*). Na classe *VeterinarioDAO* é necessário realizar as importações de classes ilustradas na Figura 40.

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import cefet.model.Veterinario;
```

Figura 40. Importações realizadas na classe *VeterinarioDAO*.

Para que seja possível lançar uma exceção específica referente às classes do pacote *dao*, a classe *ExceptionDAO* foi criada. Essa classe herda as características da classe *Exception*. O código da classe *ExceptionDAO* é exibido na Figura 41.

```
public class ExceptionDAO extends Exception{

        public ExceptionDAO(String mensagem){
            super(mensagem);
        }
}
```

Figura 41. Código da classe Exception*DAO*.

O próximo método a ser criado na classe *VeterinarioDao* é o método *alterarVeterinario.* Esse método possibilita a alteração de um registro no banco de dados. A Figura 42 exibe o código do método *alterarVeterinario*.

```
public void alterarVeterinario(Veterinario veterinario) throws ExceptionDAO{

        String sql = "update veterinario set nome=?, crvm=?, formacao=? where id=?";
        PreparedStatement stmt = null;
        Connection connection = null;
        try {
            connection= new ConexaoBancoDeDados().getConnection();
            stmt = connection.prepareStatement(sql);
            stmt.setString(1, veterinario.getNome());
            stmt.setString(2, veterinario.getCRVM());
            stmt.setString(3, veterinario.getFormacao());
            stmt.setInt(4, veterinario.getId_veterinario());
            stmt.execute();

        } catch (SQLException e) {
            e.printStackTrace();
            throw new ExceptionDAO("Erro ao alterar veterinario: "+ e);

        } finally {
            try {if (stmt != null) {stmt.close();}
            } catch(SQLException e){e.printStackTrace();}
            try {if (connection != null){connection.close();}
            } catch (SQLException e) {e.printStackTrace();}
        }
}
```

Figura 42. Método *alterarVeterinario* da classe *VeterinárioDAO*.

Em seguida, crie o método *excluirVeterinario* na classe *VeterinarioDao*. Esse método possibilita a exclusão de um registro no banco de dados. A Figura 43 exibe o código do método *excluirVeterinario*.

```
public void excluirVeterinario(int id) throws ExceptionDAO{

        String sql = "delete from veterinario where id=?";
        PreparedStatement stmt = null;
        Connection connection = null;
        try {
                connection= new ConexaoBancoDeDados().getConnection();
                stmt = connection.prepareStatement(sql);
                stmt.setInt(1, id);
                stmt.execute();

        } catch (SQLException e) {
                e.printStackTrace();
                throw new ExceptionDAO("Erro ao excluir veterinário: "+ e);

        } finally {
                try {if (stmt != null) {stmt.close();}
                } catch (SQLException e) {e.printStackTrace();}
                try {if (connection != null) {connection.close();}
                } catch (SQLException e) {e.printStackTrace();}
        }
}
```

Figura 43. Método *excluirVeterinario* da classe *VeterinárioDAO*.

Por último, crie o método listarVeterinario na classe *VeterinarioDao.* Esse método permite listar todos os campos dos registros de veterinários existentes no banco de dados. A Figura 44 exibe o código do método *listarVeterinario*.

```
 public ArrayList<Veterinario> listarVeterinario() throws ExceptionDAO{
     ResultSet rs =null;
     Connection conn=null;
     PreparedStatement stmt = null;
     Veterinario veterinario =null;
     ArrayList<Veterinario> listaDeVeterinarios = null;
try { String sql="select * from veterinario";
      conn=new ConexaoBancoDeDados().getConnection();
      stmt=conn.prepareStatement(sql);
      rs=stmt.executeQuery();
      if(rs != null){
           listaDeVeterinarios = new ArrayList<>();
             while(rs.next()){
               veterinario= new Veterinario();
               veterinario.setId_veterionario(rs.getInt("id"));
               veterinario.setNome(rs.getString("nome"));
               veterinario.setCRVM(rs.getString("crvm"));
               veterinario.setFormacao(rs.getString("formacao"));
              listaDeVeterinarios.add(veterinario);} }
     } catch (SQLException e) { e.printStackTrace();
             throw new ExceptionDAO("Erro ao listar veterinario: "+ e);
         }finally{try {if(rs != null){rs.close();}
             } catch (SQLException e) {e.printStackTrace();}
             try {if(stmt != null){stmt.close();}
             } catch (SQLException e) {e.printStackTrace();
             }try {if(conn != null){ conn.close();}}catch (Exception e)
                 e.printStackTrace();}
}return listaDeVeterinarios;}
```

Figura 44. Método *listarVeterinario* da classe *VeterinárioDAO*.

A Figura 45 exibe a estrutura de classes, após a criação dos pacotes com as suas respectivas classes.

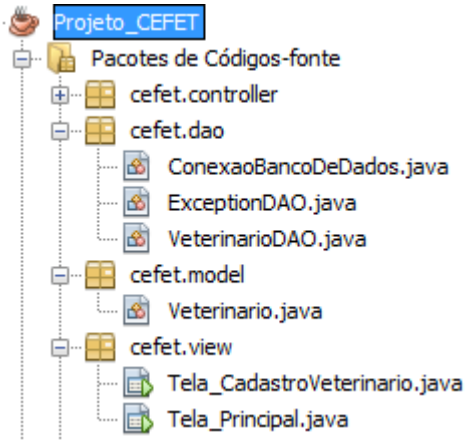


Figura 45. Pacotes e classes criadas no NetBeans.

Nessa aula, apenas uma parte do modelo MVC foi implementado. Para testar as funcionalidades das classes do pacote *dao,* crie uma classe provisória que possui o método *main* para executar testes dos métodos criados. A Figura 46. a) exibe o código criado para testar os métodos *cadastrarVeterinario* e *listarVeterinario.* A Figura 46. b) mostra a saída após a execução da classe *Teste,*

```
public class Teste {
   public static void main(String[] args){
      try {
          //Teste inserção
           Veterinario veterinario = new Veterinario("Mariah Alexandre", "876456-9","Residente");
           VeterinarioDAO veterinarioDAO= new VeterinarioDAO();
           veterinarioDAO.cadastrarVeterinario(veterinario);
           System.out.println("Cadastro efetuado.");

           //Teste listar veterinário
           ArrayList<Veterinario> listaVeterinario = veterinarioDAO.listarVeterinario();
           Iterator<Veterinario> iterator= listaVeterinario.iterator();
           System.out.println("Lista dos veterinários:");
           while(iterator.hasNext()){
               System.out.println(iterator.next().getNome());
           }
       } catch (Exception e) {
           e.printStackTrace();
       }
    }
```

a)

```
Saída - Projeto_CEFET (run)
run:
Cadastro efetuado.
Lista dos veterinários:
Roberto Tavares
Mariah Alexandre
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

b)

Figura 46. a) Código da classe *Teste*. Figura 46. b) Saída após a execução da classe *Teste*.

Aula 5

Implementação do padrão MVC
Método Cadastrar

5. Implementação do padrão MVC – Método Cadastrar

Nesta aula, será finalizado a implementação do modelo MVC iniciado na aula anterior. Será demonstrado como implementar a camada de *Controller, Model e View* para que o cadastro de veterinário seja realizado segundo o modelo MVC. A classe *Veterinário* do pacote *model* que foi criada na aula anterior será modificada para que o modelo MVC seja implementado. A classe *VeterinarioDAO* do pacote *dao* já foi implementada na íntegra, sendo assim, nenhuma alteração será necessária.

Quando o usuário acionar o botão *Cadastrar* na tela de cadastro de veterinário, a classe *Tela_CadastroVeterinario* (*view*) irá recuperar os dados preenchidos pelo usuário no formulário de cadastro e enviá-los para a controladora. Na controladora, um objeto do tipo *Veterinario* é instanciado e encaminhado para a camada *model* (*Veterinario*). Uma vez que a *model* receber o objeto do tipo *Veterinario* enviado pela controladora, ela irá chamar um método da classe *dao* responsável pela inserção no banco de dados.

O primeiro passo para finalizar o modelo MVC será criar o método *cadastrarVeterinario* na classe *Veterinario*. Esse método receberá como parâmetro um objeto do tipo *Veterinário* e instanciará um objeto do tipo *VeterinarioDAO.* Através dessa instancia, o método *cadastrarVeterinario* pertencente a classe *VeterinarioDAO* é chamado para inserção no banco de dados. A Figura 47 mostra o código do método *cadastrarVeterinario* criado na classe *Veterinario*. Note que é necessário importar a classe *VeterinarioDAO*.

```
public void cadastrarVeterinario(Veterinario veterinario) throws ExceptionDAO{
        new VeterinarioDAO().cadastrarVeterinario(veterinario);
    }
```

Figura 47. Código do método *cadastrarVeterinario* da classe *Veterinario*.

O próximo passo é criar a classe *VeterinarioController* no pacote

controller. Após a criação da classe, crie o método *cadastrarVeterinario* cujo código é ilustrado na Figura 48. Esse método receberá como parâmetro os dados necessários para cadastramento de um veterinário (nome, CRVM e formação). Repare que esses dados serão enviados pela classe *Tela_CadastroVeterinario (view).*

```
public void cadastrarVeterinario(String nome, String CRVM, String formacao) throws Exception{
    if(nome!=null && nome.length() > 0 && CRVM !=null && CRVM.length() > 0
    && formacao.length() > 0 && formacao != null){
        Veterinario veterinario = new Veterinario(nome,CRVM,formacao);
        veterinario.cadastrarVeterinario(veterinario);
    }else{
    throw new Exception( "Preencha os campos corretamente");
    }
}
```

Figura 48. Código do método *cadastrarVeterinario* da classe *VeterinarioController*.

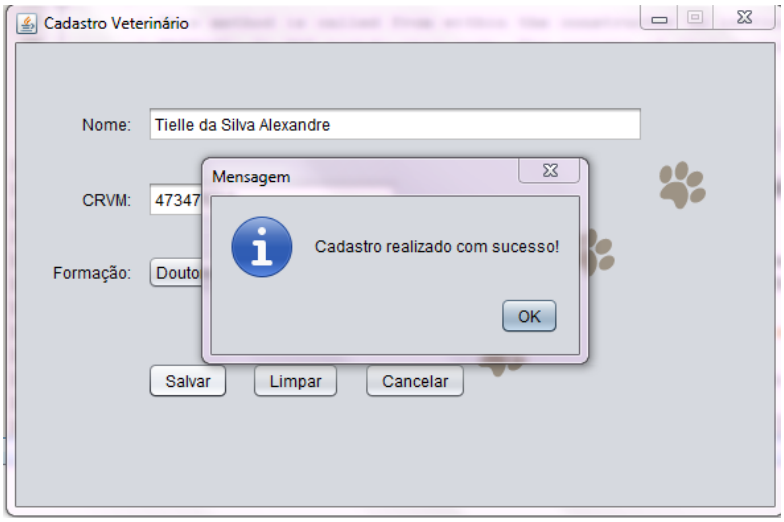
Uma verificação é realizada no método *cadastrarVeterinario* a fim de checar se os dados foram devidamente preenchidos. Caso algum dado não esteja preenchido pelo usuário, uma exceção será lançada. Por fim, a controladora envia o objeto para a classe *Veterinario* (*model*).

A última alteração necessária ocorrerá na classe *Tela_CadastroVeterinario* (*view*) no método *salvar_CadastroVeterinario*. Esse método é acionado quando o usuário clicar no botão *Salvar* do formulário. Através dos métodos *getText* e *getSelectedItem()*, os dados preenchidos pelo usuário são recuperados para serem enviados para a controladora. A Figura 49 exibe o código do método salvar_*CadastroVeterinario*.

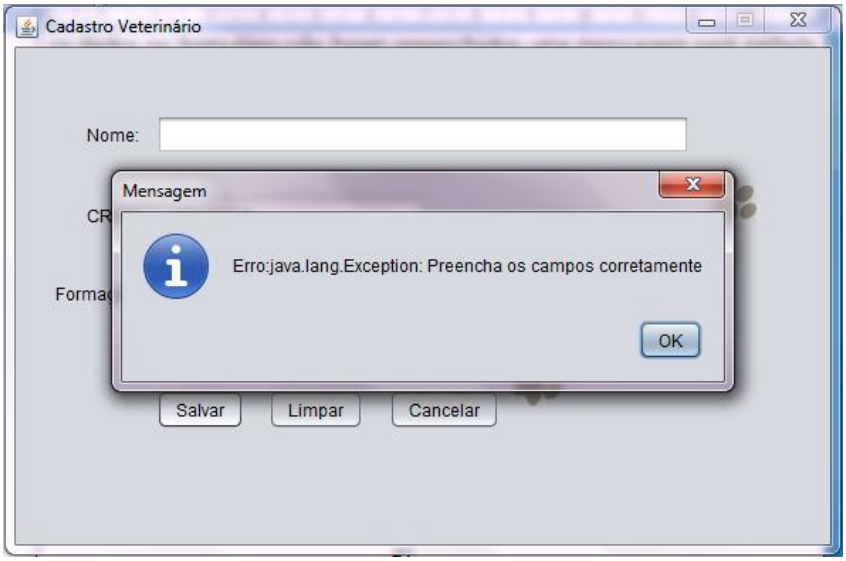
```
private void salvar_CadastroVeterinario(java.awt.event.ActionEvent evt) {
    //Salva os dados referentes a um veterinário.
   try{
     VeterinarioController veterinarioController = new VeterinarioController();
     veterinarioController.cadastrarVeterinario(JTextNome.getText(), JTextCRVM.getText(),
     JComboFormação.getSelectedItem().toString());
     JOptionPane.showMessageDialog(null,"Cadastro realizado com sucesso!");
   } catch (Exception e) {
             JOptionPane.showMessageDialog(null,"Erro:" + e);
    }
}
```

Figura 49. Código do método *salvar_CadastroVeterinario* da classe *Tela_CadastroVeterinario*.

Uma mensagem de sucesso será exibida para usuário caso o cadastro de veterinário seja realizado no banco de dados. Se algum erro ocorrer, uma mensagem de erro será mostrada ao usuário. Por exemplo, se os dados no formulário não forem preenchidos uma mensagem será exibida para comunicar ao usuário.



a)



b)

Figura 50. a) Mensagem de sucesso exibida ao usuário. Figura 50. b) Mensagem de erro exibida ao usuário se os dados no formulário não forem preenchidos.

Implementação do padrão MVC
Método Consultar

6. Implementação do padrão MVC – Método Consultar

Nesta aula, será implementada o método consultar segundo o padrão MVC. A construção de uma tela de consulta de veterinário será demonstrada com os principais componentes gráficos utilizados. Além disso, será explicado o código das classes do padrão MVC para a implementação do método consultar. No exemplo dessa aula, a consulta dos registros será exibida em uma tabela e se caso o usuário der um clique duplo em uma das linhas dessa tabela, os valores da linha selecionada serão carregados em outra tela.

A *Tela_ConsultaVeterinario* que será utilizada nesta aula é exibida na Figura 51. a). Note que quase todos os componentes já foram apresentados na aula 3 dessa apostila, com exceção do componente gráfico denominado de *JTable*. Essa tela de consulta de veterinário é acionada pela *Tela Principal* através de um clique no menu Consulta→Veterinário (Figura 51. b).

a)

b)

Figura 51. a) Tela de consulta de veterinário. Figura 51. b) Tela Principal exibindo o menu Consulta.

O código para a implementação do evento que inicia a *Tela_ConsultaVeterinario* é exibido na Figura 52. Repare que o objeto da *Tela Principal* é passado como parâmetro do construtor da tela de consulta de veterinário.

```
private void abrir_TelaConsultaVeterinario(java.awt.event.ActionEvent evt) {
    Tela_ConsultaVeterinario tela_consulta = new Tela_ConsultaVeterinario(this);
    tela_consulta.setVisible(true);
    setVisible(false);
}
```

Figura 52. Código do evento da *Tela_Principal* que abre a *Tela_ConsultaVeterinario*.

O componente *JTable* representa uma tabela e permite que os dados possam ser exibidos e/ou editados em uma estrutura composta por linhas e

colunas. Nessa aula, o componente *JTable* será usado apenas para exibir os dados armazenados no banco de dados

Para inserir o componente *JTable* em um janela, clique na paleta de componentes gráficos do NetBeans e escolha o componente chamado de *Tabela* em *Controles Swing*. A Figura 53 mostra a tabela inserida e posicionada no local desejado na tela de consulta.

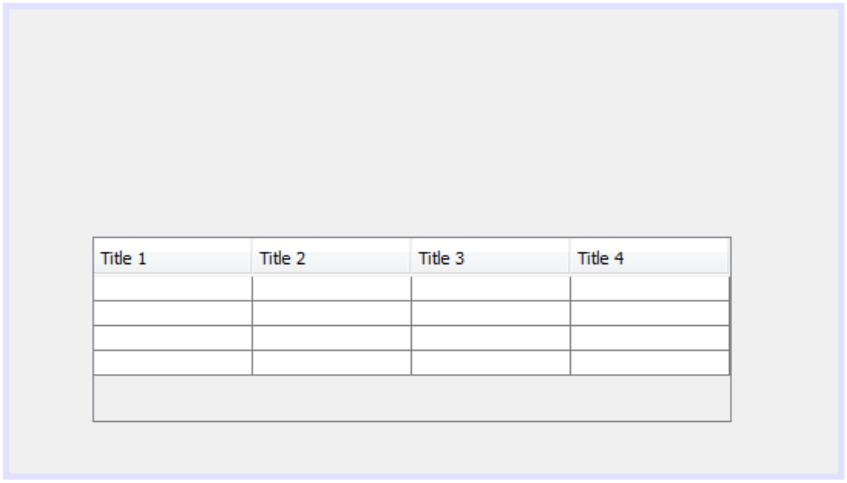


Figura 53. Componente *JTable* inserido na janela.

Para que o componente *JTable* fique com a aparência da Figura 51, é necessário que as seguintes modificações sejam realizadas nas propriedades desse componente:

1. Altere o nome da variável de acordo com o padrão de nomes estabelecido para os componentes. Nesse exemplo, o nome da variável foi alterado para *JTableConsultaVeterinario;*
2. Altere as linhas e as colunas que serão visualizadas na tabela. Para isso, clique na propriedade chamada de *model* do componente *JTable* para exibir a tela que permite modificar as linhas e colunas de uma tabela(Figura 54).

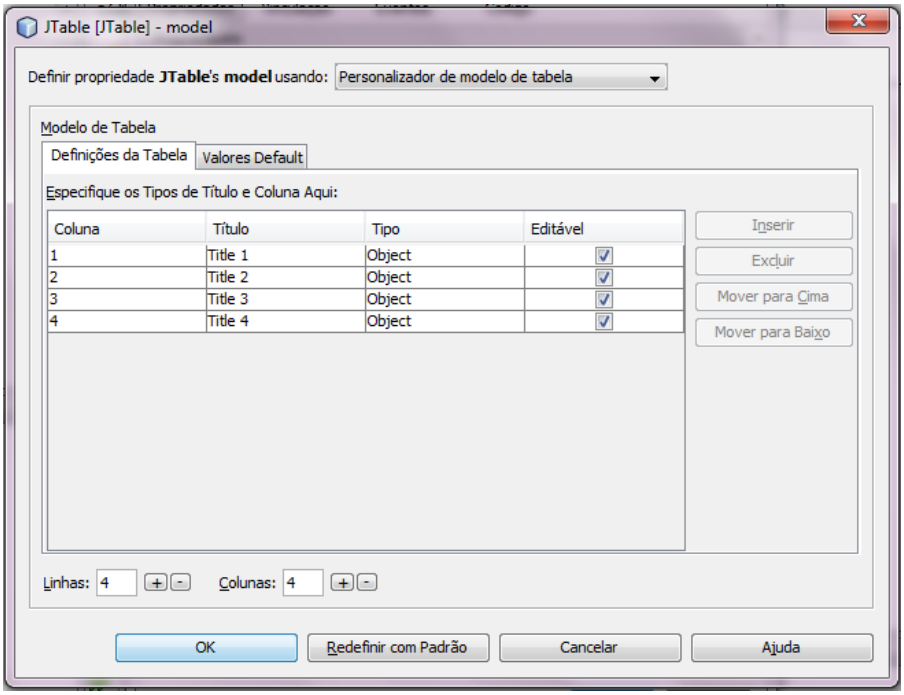


Figura 54. Tela de configuração da propriedade *model* do componente *JTable.*

Na coluna *Título*, é possível alterar o nome das colunas da tabela; na coluna *Tipo* altera-se o tipo de dado (*Object*, String, *Boolean*) que será armazenado em uma célula e na coluna *Editável* é possível habilitar e desabilitar a edição de uma determinada célula da tabela.

No canto inferior da inferior da tela, pode-se aumentar e diminuir os números de linhas e colunas da tabela. Além disso, a ordem da disposição das colunas pode ser alterada utilizando os botões *Mover para Cima ou para Baixo.* Para o componente *JTable* do exemplo (Figura 51), a tela de configuração da propriedade *model* foi alterada conforme a Figura 55. Note que as células foram desabilitadas para a edição dos dados e que a tabela, inicialmente, não irá possuir nenhuma linha.

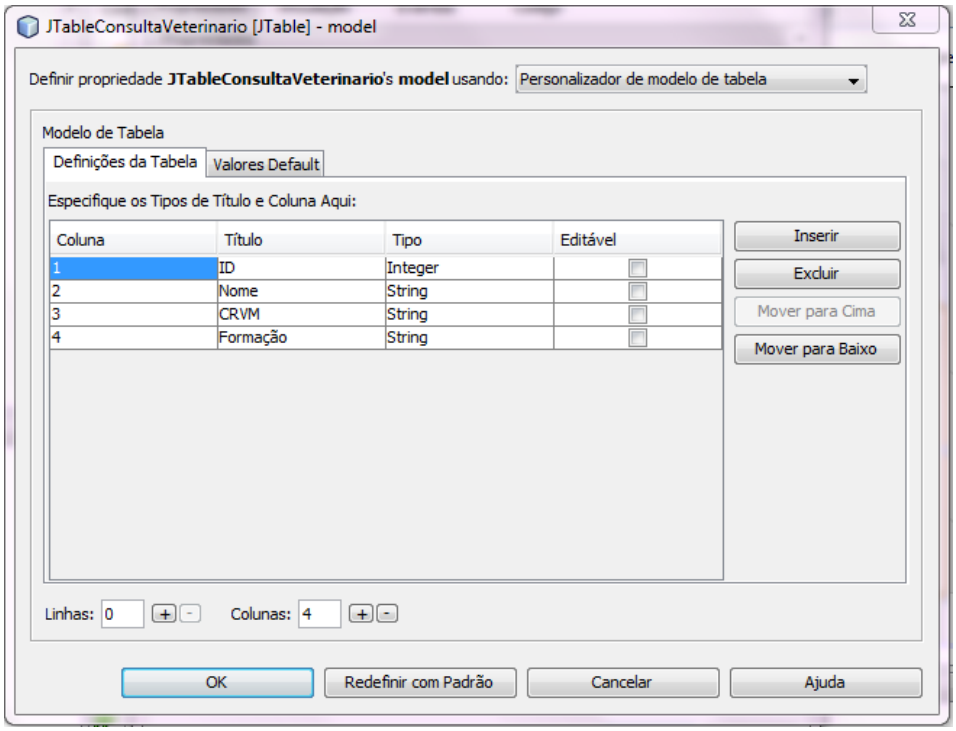


Figura 55. Tela de configuração da propriedade *model* do componente *JTable* implementado no exemplo.

A *Tela_ConsultaVeterinario* possui o seguinte funcionamento: o usuário informa um nome completo ou parte dele e quando esse clicar no botão *Buscar* (lupa), a tela irá exibir em uma tabela a lista de registros de veterinários ordenados por nome que possuem o parâmetro informado em qualquer posição. Caso o usuário clique duas vezes em umas das linhas exibidas na tabela, os valores da linha selecionada serão carregados na *Tela_CadastroVeterinario.*

O código do evento da *Tela_ConsultaVeterinario* que consulta os registros de veterinários no banco de dados é exibido na Figura 56. Esse evento é acionado quando o usuário clicar no botão *Buscar.* Um objeto do tipo *VeterinarioController* (controller) é instanciado para que o método *listarVeterinario* seja chamado passando como parâmetro o nome informado pelo usuário. O *DefaultTableModel* é a implementação padrão de *TableModel* e é utilizado quando nenhum modelo é especificado.

```
private void consultar_Veterinario(java.awt.event.ActionEvent evt) {
  String nome = jTextNome.getText();
   try{
       DefaultTableModel tableModel =(DefaultTableModel) JTableConsultaVeterinario.getModel();

       VeterinarioController veterinarioController= new VeterinarioController();
       ArrayList<Veterinario> listaVeterinario =veterinarioController.listarVeterinario(nome);

       Iterator<Veterinario> iterator= listaVeterinario.iterator();
       while(iterator.hasNext()){
        Veterinario veterinario=iterator.next();
         tableModel.addRow(new Object[]{veterinario.getId_veterinario(),veterinario.getNome(),
         veterinario.getCRVM(),veterinario.getFormacao()});
        }
    } catch (Exception e) {
       JOptionPane.showMessageDialog(null,"Erro:" + e);
    }
```

Figura 56. Código do método *consultar_Veterinario* da *Tela_ConsultaVeterinario* .

Após a execução do método *listarVeterinario*, a lista de veterinários retornada por esse método é percorrida e os valores são adicionados ao *JTable* através do método *addRow*. Note que cada objeto do tipo Veterinario pertencente à lista é transformado em uma linha da tabela.

Outro método que deve ser implementado na *Tela_ConsultaVeterinario* é aquele que recupera os valores de uma linha selecionada pelo usuário dentre as linhas exibidas na tabela . Os valores recuperados são enviados para a Tela_CadastroVeterinario através do método construtor. O código desse método (*mouseClicked*) é ilustrado na Figura 57.

```
JTableConsultaVeterinario.addMouseListener(new MouseAdapter(){
     @Override
public void mouseClicked(MouseEvent e){
 if(e.getClickCount() == 2){
   Integer id=(Integer)JTableConsultaVeterinario.getModel().getValueAt(JTableConsultaVeterinario.getSelectedRow(), 0);
   String nome=(String)JTableConsultaVeterinario.getModel().getValueAt(JTableConsultaVeterinario.getSelectedRow(), 1);
   String crvm=(String)JTableConsultaVeterinario.getModel().getValueAt(JTableConsultaVeterinario.getSelectedRow(), 2);
   String formacao=(String)JTableConsultaVeterinario.getModel().getValueAt(JTableConsultaVeterinario.getSelectedRow(), 3);

 Tela_CadastroVeterinario telaCadastro = new Tela_CadastroVeterinario(id, nome, crvm, formacao,telaPrincipal);
 telaCadastro.setVisible(true);
 setVisible(false);

 }}
 });
```

Figura 57. Código do método *mouseClicked* da Tela_ConsultaVeterinario.

Na *Tela_ConsultaVeterinario* deve ser declarado um atributo privado do tipo *Tela_Principal* e um construtor com parâmetros (Figura 58. a). As

importações de classes realizadas nessa tela são mostradas na Figura 58. b).

```
private Tela_Principal telaPrincipal;

public Tela_ConsultaVeterinario(Tela_Principal telaPrincipal){
  this.telaPrincipal=telaPrincipal;
  initComponents();
}
```

a)

```
import cefet.controller.VeterinarioController;
import cefet.model.Veterinario;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.ArrayList;
import java.util.Iterator;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
```

b)

Figura 58. a) Atributo e método construtor da *Tela_ConsultaVeterinario*. Figura 58. b) Importações de classes da *Tela_ConsultaVeterinario*.

Na classe *VeterinarioController* (camada *controller*), o método *listarVeterinario* deve ser implementado conforme o código exibido na Figura 59.

```
public ArrayList<Veterinario> listarVeterinario(String nome) throws ExceptionDAO{
      Veterinario veterinario = new Veterinario();
      return veterinario.listarVeterinario(nome);
  }
```

Figura 59. Código do método *listarVeterinario* da classe *VeterinarioController*.

Na classe *Veterinario* (camada *model*), o método *listarVeterinario* deve ser implementado conforme o código exibido na Figura 60.

```
public ArrayList<Veterinario> listarVeterinario(String nome) throws ExceptionDAO{
    return new VeterinarioDAO().listarVeterinario(nome);
}
```

Figura 60. Código do método *listarVeterinario* da classe *Veterinario*.

Por fim, o método *listarVeterinario* da classe *VeterinarioDAO* (camada *dao*) deve ser modificado conforme o código mostrado na Figura 61. O comando SQL foi alterado para que o método retorne todos os registros de veterinário do banco de dados ordenados por nome e que possuem em qualquer posição uma string igual à informada pelo usuário. O método *listarVeterinario* recebe agora o parâmetro nome.

```
 public ArrayList<Veterinario> listarVeterinario(String nome) throws ExceptionDAO{
     ResultSet rs =null;
     Connection conn=null;
     PreparedStatement stmt = null;
     ArrayList<Veterinario> listaDeVeterinarios = null;
try { String sql="select * from veterinario where nome like '%"+nome+"%' order by nome";
      conn=new ConexaoBancoDeDados().getConnection();
      stmt=conn.prepareStatement(sql);
      rs=stmt.executeQuery();
      if(rs != null){
           listaDeVeterinarios = new ArrayList<>();
             while(rs.next()){
               Veterinario veterinario= new Veterinario();
               veterinario.setId_veterinario(rs.getInt("id"));
               veterinario.setNome(rs.getString("nome"));
               veterinario.setCRVM(rs.getString("crvm"));
               veterinario.setFormacao(rs.getString("formacao"));
              listaDeVeterinarios.add(veterinario);} }
     } catch (SQLException e) { e.printStackTrace();
             throw new ExceptionDAO("Erro ao listar veterinario: "+ e);
         }finally{try {if(rs != null){rs.close();}
             } catch (SQLException e) {e.printStackTrace();}
             try {if(stmt != null){stmt.close();}
             } catch (SQLException e) {e.printStackTrace();
             }try {if(conn != null){ conn.close();}}catch (Exception e) {
                 e.printStackTrace();}
}return listaDeVeterinarios;}
```

Figura 61. Código do método *listarVeterinario* da classe *VeterinarioDAO*.

A Figura 62 ilustra a *Tela_ConsultaVeterinario* exibindo uma consulta referente aos veterinários que possuem o nome “A*na*” em qualquer posição.

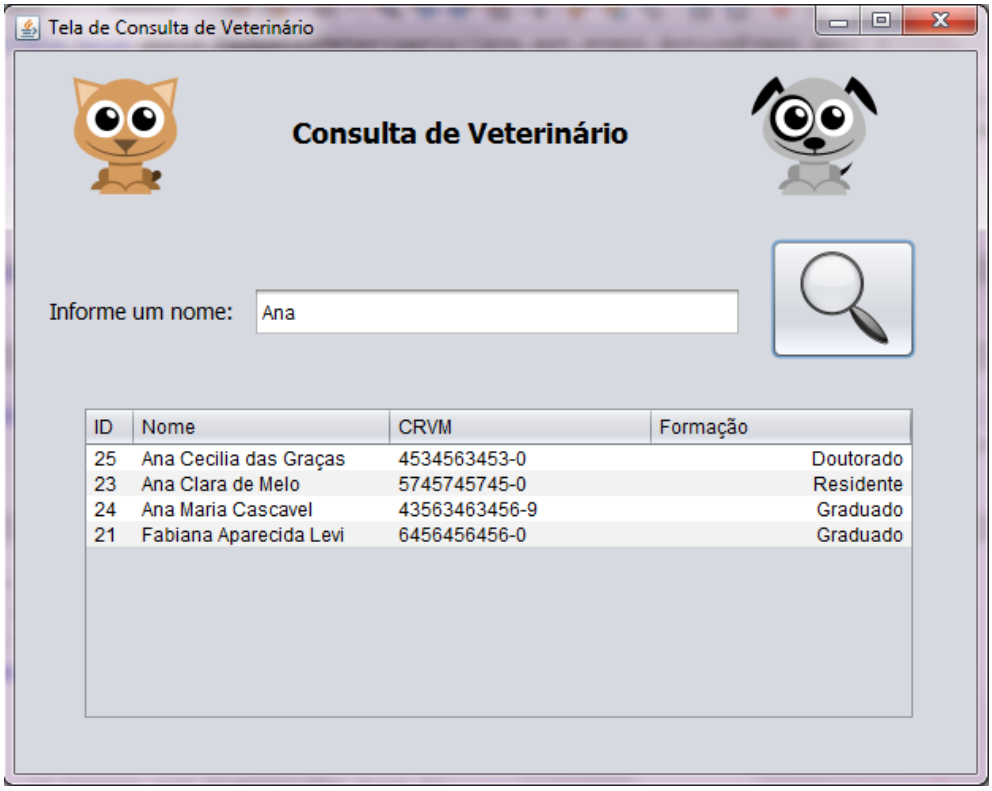


Figura 62. *Tela_ConsultaVeterinario* exibindo uma consulta.

Quando o usuário der um clique duplo em uma determinada linha, os valores dessa linha serão carregados na *Tela_CadastroVeterinario* (Figura 63. a). Para isso, declare o código do construtor e atributos mostrados na Figura 63. b) na *Tela_CadastroVeterinario*.

a)

```
private Tela_Principal telaprincipal;
private int idVeterinario;

public Tela_CadastroVeterinario(Integer id, String nome, String CRVM, String formacao, Tela_Principal telaPrincipal){
initComponents();
this.telaprincipal=telaPrincipal;
JTextNome.setText(nome);
JTextCRVM.setText(CRVM);
this.idVeterinario=id;

for(int n=0; n < JComboFormação.getItemCount(); n=n+1){
  if(JComboFormação.getItemAt(n).equals(formacao)){
     JComboFormação.setSelectedIndex(n);
  }}

}
```

b)

Figura 63. a) *Tela_CadastroVeterinario* exibindo os valores carregados de uma linha da tabela (*JTable*). Figura 63. b) Atributos e o método construtor da *Tela_CadastroVeterinario*.

Implementação do padrão MVC - Método Excluir e Alterar

7. Implementação do padrão MVC – Método Excluir e Alterar

Nesta aula, serão implementados os métodos para a exclusão e alteração segundo o padrão MVC. A tela de cadastro de veterinário será modificada possibilitando a exclusão ou alteração de um registro de veterinário armazenado no banco de dados. Primeiramente, o método consultar deverá ser executado para que o registro desejado seja escolhido para a exclusão ou alteração.

O primeiro passo será modificar a tela de cadastro de veterinário adicionando mais dois botões. Um botão irá disparar o método de alteração e outro botão, o método de exclusão. A Figura 64 ilustra a tela com as devidas modificações. Após a execução de uma operação de consulta, a tela de cadastro de veterinário foi acionada carregando os dados de um registro selecionado pelo usuário. Repare que neste caso, o botão *Salvar* deve aparecer desabilitado, já que apenas as operações para a exclusão ou alteração serão permitidas.

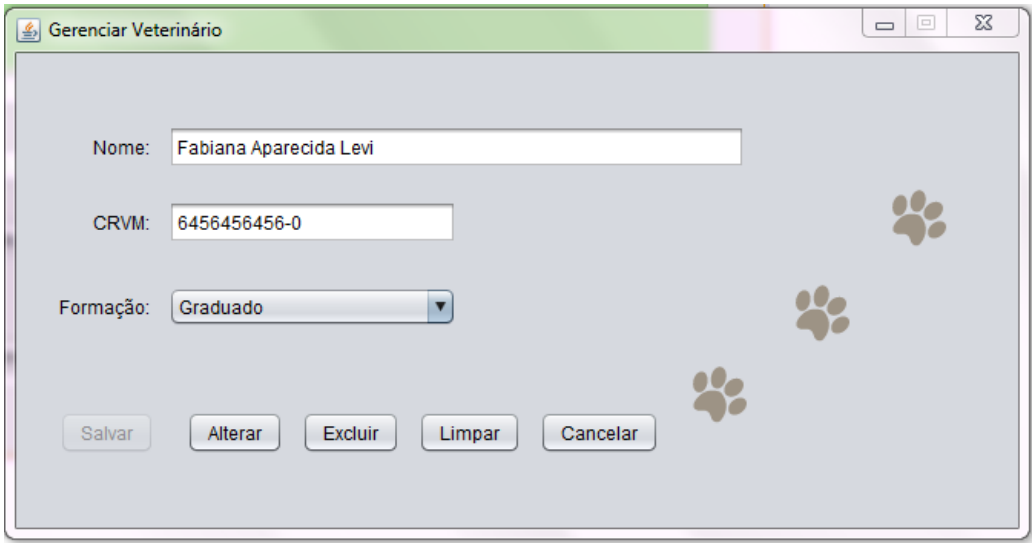


Figura 64. Tela de cadastro de veterinário com acréscimo dos botões para *Alterar* e *Excluir*.

Neste caso, será necessário alterar o construtor da tela de cadastro de veterinário chamado no evento *mouseClicked* na tela de consulta de veterinário. Após os dados serem carregados na tela de cadastro, os botões terão a propriedade *Enabled* alterada, conforme o código exibido na Figura 65.

```
public Tela_CadastroVeterinario(Integer id, String nome, String CRVM, String formacao, Tela_Principal telaPrincipal){
initComponents();
this.telaprincipal=telaPrincipal;
JTextNome.setText(nome);
JTextCRVM.setText(CRVM);
this.idVeterinario=id;

for(int n=0; n < JComboFormação.getItemCount(); n=n+1){
  if(JComboFormação.getItemAt(n).equals(formacao)){
     JComboFormação.setSelectedIndex(n);
  }}

  JButtonAlterar.setEnabled(true);
  JButtonExcluir.setEnabled(true);
  JButtonSalvar.setEnabled(false);
}
```

Figura 65. Método construtor da tela de cadastro de veterinário alterada.

Quando a tela de cadastro de veterinário for disparada pelo menu *Cadastro* da tela principal, os botões *Alterar* e *Excluir* devem aparecer desabilitados, conforme ilustrado na Figura 66. O título da tela foi alterado para *Gerenciar Veterinário* já que a mesma tela permitirá as operações de salvar, excluir e alterar os dados de um veterinário. Sendo assim, a tela de cadastro de veterinário será agora chamada de tela de gerenciar veterinário.

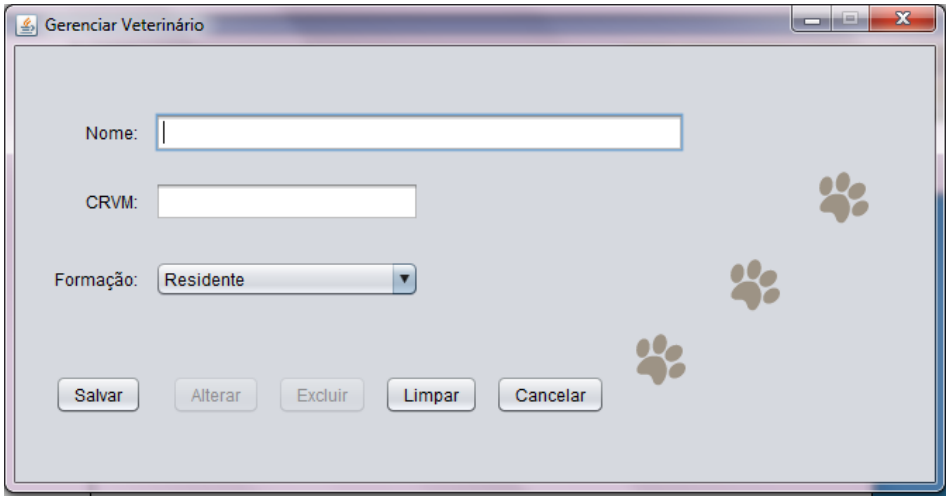


Figura 66. Tela acionada pelo menu *Cadastro* da tela principal.

O próximo passo é implementar a ação do botão *Alterar* da tela de gerenciar veterinário. O código do método *alterar_CadastroVeterinario* é exibido na Figura 67. Nesse método, um objeto do tipo *VeterinarioController* é instanciado e depois o método *alterarVeterinario* dessa classe é chamado. Como parâmetro, esse método recebe os valores informados pelo usuário na tela de gerenciar veterinário.

Se a operação obtiver êxito, uma mensagem de sucesso será exibida. Caso o contrário, uma mensagem com o respectivo erro será exibida para usuário. Após a operação de alteração, a tela de gerenciar veterinário deverá ser fechada e a tela principal deverá ser exibida para o usuário.

```
private void alterar_CadastroVeterinario(java.awt.event.ActionEvent evt) {
    try {
        VeterinarioController veterinarioController = new VeterinarioController();
        veterinarioController.alterarVeterinario(this.idVeterinario,JTextNome.getText(),
        JTextCRVM.getText(), JComboFormação.getSelectedItem().toString());
        JOptionPane.showMessageDialog(null,"O cadastro foi alterado com sucesso!");
        this.setVisible(false);
        telaprincipal.setVisible(true);
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null,"Erro:" + e);
    }
}
```

Figura 67. Código do método *alterar_CasdastroVeterinario* na tela de gerenciar veterinário.

Na classe *VeterinarioController* (camada *Controller*), crie o método *alterarVeterinario* que recebe como parâmetros os dados informados pelo usuário na tela de gerenciar veterinario. Nesse método, uma verificação é feita a fim de garantir que todos os dados estejam preenchidos. O código desse método é ilustrado na Figura 68.

```
public void alterarVeterinario(int id,String nome, String CRVM, String formacao)throws Exception{

    if(nome!=null && nome.length() > 0 && CRVM !=null && CRVM.length() > 0 && formacao.length() > 0 && formacao != null){
      Veterinario veterinario = new Veterinario(id,nome,CRVM,formacao);
      veterinario.alterarVeterinario(veterinario);
    } else {
      throw new Exception( "Preencha os campos corretamente");
    }
}
```

Figura 68. Código do método *alterarVeterinario* na classe *VeterinarioController*.

Por sua vez, crie o método *alterarVeterinario* na classe *Veterinario* pertencente à camada *model*. O código desse método é exibido na Figura

69. O código do método *alterarVeterinario* da camada *dao* foi apresentado na aula 4.

```
public void alterarVeterinario(Veterinario veterinario)throws ExceptionDAO{
        new VeterinarioDAO().alterarVeterinario(veterinario);
}
```

Figura 69. Código do método *alterarVeterinario* na classe *Veterinario*.

O próximo passo é implementar a ação do botão *Excluir* que será executada quando o usuário pressionar o botão. O código desse método é mostrado na Figura 70. Nesse método, um objeto do tipo *VeterinarioController* é instanciado e depois o método *excluirVeterinario*

```
private void excluir_Veterinario(java.awt.event.ActionEvent evt) {
    try {
        VeterinarioController veterinarioController = new VeterinarioController();
        veterinarioController.excluirVeterinario(this.idVeterinario);
        JOptionPane.showMessageDialog(null,"Exclusão realizada com sucesso!");
        this.setVisible(false);
        this.telaprincipal.setVisible(true);
     } catch (ExceptionDAO | HeadlessException e) {
        JOptionPane.showMessageDialog(null,"Erro:" + e);
     }
}
```

dessa classe é chamado. Como parâmetro, esse método recebe o valor do atributo *idVeterinario.* Da mesma forma, uma mensagem será exibida ao usuário informando sobre o êxito ou não da operação de exclusão.

Figura 70. Código do método *excluirVeterinario* na tela de gerenciar veterinário.

```
public void excluirVeterinario(int id)throws ExceptionDAO{
        Veterinario veterinario = new Veterinario();
        veterinario.excluirVeterinario(id);
}
```

a classe *VeterinarioController* (camada *Controller*), crie o método *excluirVeterinario,* conforme o código exibido na Figura 71.

Figura 71. Código do método *excluirVeterinario* na classe *VeterinarioController*.

Por fim, crie o método *excluirVeterinario* na classe *Veterinario* pertencente à camada *model*, conforme o código ilustrado na Figura 72. O código do método *excluirVeterinario* da camada *dao* foi apresentado na aula 4.

```
public void excluirVeterinario(int id)throws ExceptionDAO{
        new VeterinarioDAO().excluirVeterinario(id);
}
```

Figura 72. Código do método *excluirVeterinario* na classe *Veterinario*.

4) Referências Bibliográficas

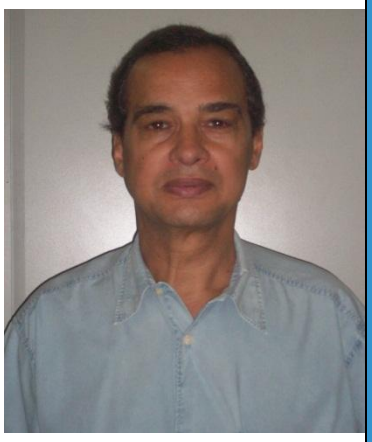
Chen, P., Pin-Shan. (1977). *The Entity-Relationship Model*.

Gamma, Erich. Helm, Richard. Johnson, Ralph. Vlissides, John. Padrões de Projeto: Soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2000.

Heuser, C., Alberto. (1998). Projeto de Banco de Dados. Editora: Sagra-Luzzatto.

NetBeans IDE. Disponível em: < https://netbeans.org/index.html>. Acesso em 09/02/2016.

COM A PALAVRA, O COORDENADOR GERAL ...
Professor D. Sc. Mauro Godinho Gonçalves



A implementação de curso a distância sugere como requisito de desenvolvimento o conhecimento da tecnologia digital e das ferramentas que possibilitam a interação entre seus interlocutores.

Nesta obra sobre educação tecnológica, são discutidas as diversas possibilidades que a educação a distância oferece e apresentada a importância do mundo digital para sua implementação. Discute-se, também, as tecnologias da informação e comunicação e as várias redes sociais muito utilizadas na comunicação entre as pessoas atualmente.

O autor deste trabalho é professor com reconhecida competência nesta área, lecionando no ensino técnico, superior e pós. Assim, espero que apreciem o seu trabalho.