Projeto Pets

Nomes: Angely Zambrano, Cibely Godoy Martins, Guilherme Silva Krast, Juliana Xavier, Lucas Moro

Nome: Lucas

1. Classe Pet

Esta classe serve como um modelo de dados (entidade) para representar um animal de estimação. Ela contém apenas atributos para armazenar os dados e os métodos para acessá-los e modificá-los (getters e setters).

Construtores

public Pet()

- Descrição: Este é o construtor padrão (ou sem argumentos). Ele cria uma instância da classe Pet com todos os seus atributos inicializados com os valores padrão (null para objetos como String, e 0 para tipos primitivos numéricos como int).
- Uso: É útil em situações onde um objeto Pet precisa ser criado primeiro e seus dados preenchidos depois, utilizando os métodos set.

public Pet(int id_pet, String nome, ...)

- Descrição: Este é um construtor parametrizado que permite criar uma instância da classe Pet e inicializar todos os seus atributos de uma só vez, no momento da criação.
- Parâmetros:
    - id_pet: O identificador único do pet.
    - nome: O nome do pet.
    - especie: A espécie do pet (ex: "Cão", "Gato").
    - raca: A raça específica do pet.
    - data_nascimento: O ano de nascimento do pet.
    - peso: O peso do pet em quilogramas.
    - sexo: O sexo do pet (ex: "Macho", "Fêmea").
    - id_tutor: O identificador único do tutor.
    - nomeTutor: O nome completo do tutor.
- Uso: É o construtor principal utilizado no PetController para criar um novo pet com todos os dados fornecidos pelo usuário no formulário.

Métodos Getters e Setters (Acessores e Modificadores)

Estes métodos seguem o princípio do encapsulamento, protegendo o acesso direto aos atributos da classe e permitindo a validação ou lógica adicional no futuro, se necessário.

public int getIdPet()

- Descrição: Retorna o valor do atributo id_pet.
- Retorno: (int) - O identificador único do pet.

public void setIdPet(int id_pet)

- Descrição: Define ou atualiza o valor do atributo id_pet.
- Parâmetro: id_pet - O novo identificador para o pet.

public String getNome()

- Descrição: Retorna o nome do pet.
- Retorno: (String) - O nome atual do pet.

public void setNome(String nome)

- Descrição: Define ou atualiza o nome do pet.
- Parâmetro: nome - O novo nome para o pet.

public String getEspecie()

- Descrição: Retorna a espécie do pet.
- Retorno: (String) - A espécie do pet (ex: "Cão").

public void setEspecie(String especie)

- Descrição: Define ou atualiza a espécie do pet.
- Parâmetro: especie - A nova espécie para o pet.

public String getRaca()

- Descrição: Retorna a raça do pet.
- Retorno: (String) - A raça do pet (ex: "Labrador").

public void setRaca(String raca)

- Descrição: Define ou atualiza a raça do pet.
- Parâmetro: raca - A nova raça para o pet.

public int getDataNascimento()

- Descrição: Retorna o ano de nascimento do pet.
- Retorno: (int) - O ano em que o pet nasceu.

public void setDataNascimento(int data)

- Descrição: Define ou atualiza o ano de nascimento do pet.
- Parâmetro: data - O novo ano de nascimento para o pet.

public int getPeso()

- Descrição: Retorna o peso do pet.
- Retorno: (int) - O peso do pet em kg.

public void setPeso(int peso)

- Descrição: Define ou atualiza o peso do pet.
- Parâmetro: peso - O novo peso em kg para o pet.

public String getSexo()

- Descrição: Retorna o sexo do pet.
- Retorno: (String) - O sexo do pet (ex: "Macho").

public void setSexo(String sexo)

- Descrição: Define ou atualiza o sexo do pet.
- Parâmetro: sexo - O novo sexo para o pet.

public int getIdTutor()

- Descrição: Retorna o ID do tutor do pet.
- Retorno: (int) - O identificador único do tutor.

public void setIdTutor(int id_tutor)

- Descrição: Define ou atualiza o ID do tutor do pet.
- Parâmetro: id_tutor - O novo ID do tutor.

public String getNomeTutor()

- Descrição: Retorna o nome do tutor do pet.
- Retorno: (String) - O nome completo do tutor.

public void setNomeTutor(String nomeTutor)

- Descrição: Define ou atualiza o nome do tutor do pet.
- Parâmetro: nomeTutor - O novo nome do tutor.

---

2. Classe PetController

Esta classe controla a interface gráfica e a lógica da aplicação.

Métodos do Ciclo de Vida do JavaFX

public void start(Stage primaryStage)

- Descrição: É o ponto de entrada principal da aplicação JavaFX. Este método é chamado automaticamente pelo framework para construir a interface do usuário.
- Funcionalidades:
    1. Chama carregarDados() para ler os pets salvos em arquivo.
    2. Define o título da janela principal (Stage).
    3. Cria os painéis de layout (formBox, tableBox, buttonBox) chamando os métodos de construção correspondentes.
    4. Organiza os painéis na tela usando um BorderPane.
    5. Adiciona um "ouvinte" (listener) na tabela, que chama carregarPetNoFormulario() sempre que um pet é selecionado.
    6. Monta a Scene (cena), a insere no Stage e a exibe para o usuário.
- Parâmetro: primaryStage - O "palco" ou janela principal da aplicação, fornecido pelo JavaFX.

public static void main(String[] args)

- Descrição: Ponto de entrada padrão de qualquer aplicação Java. Sua única função aqui é chamar launch(args), que inicia o ambiente JavaFX e, por sua vez, chama o método start.

Métodos de Persistência de Dados

private void salvarDados()

- Descrição: Salva a lista de pets (petList) em um arquivo binário (pets.dat). Ele serializa a lista de objetos, convertendo-os em um fluxo de bytes que pode ser gravado em disco.
- Funcionamento: Utiliza um ObjectOutputStream para escrever o objeto ArrayList<Pet> no arquivo definido pela constante ARQUIVO_DE_DADOS.

private void carregarDados()

- Descrição: Carrega a lista de pets do arquivo pets.dat ao iniciar a aplicação.
- Funcionamento:
    1. Verifica se o arquivo de dados existe. Se não, encerra a execução.
    2. Usa um ObjectInputStream para ler o fluxo de bytes do arquivo e o desserializa, reconstruindo a List<Pet>.
    3. Atualiza a petList da aplicação com os dados carregados.

4. Calcula o próximo nextId disponível, pegando o maior ID existente e somando 1, para evitar IDs duplicados.

Métodos de Ação e Lógica

private void adicionarPet()

- Descrição: É chamado quando o botão "Adicionar" é clicado. Ele coleta os dados do formulário, valida-os e cria um novo registro de pet.
- Funcionamento:
    1. Chama validarCampos(). Se a validação falhar, a função para.
    2. Lê os valores dos TextFields e do ComboBox.
    3. Cria uma nova instância de Pet usando o construtor parametrizado.
    4. Adiciona o novo pet à petList (a tabela se atualiza automaticamente).
    5. Chama limparCampos(), mostrarSucesso() e, crucialmente, salvarDados() para persistir a alteração.

private void editarPet()

- Descrição: Chamado pelo botão "Editar". Atualiza os dados de um pet já existente que foi selecionado na tabela.
- Funcionamento:
    1. Obtém o Pet selecionado na tableView. Se nenhum pet for selecionado, exibe um erro.
    2. Chama validarCampos() para os dados no formulário.
    3. Usa os métodos set do objeto Pet selecionado para atualizar seus atributos com os novos valores do formulário.
    4. Chama tableView.refresh() para garantir que a tabela reflita as mudanças.
    5. Chama limparCampos(), mostrarSucesso() e salvarDados().

private void removerPet()

- Descrição: Chamado pelo botão "Remover". Exclui o pet selecionado na tabela.
- Funcionamento:
    1. Obtém o Pet selecionado. Se nenhum for selecionado, exibe um erro.
    2. Exibe uma caixa de diálogo (Alert) de confirmação para evitar remoções acidentais.

3. Se o usuário confirmar, o pet é removido da petList.
4. Chama limparCampos(), mostrarSucesso() e salvarDados().

Métodos Utilitários e de Feedback

private void limparCampos()

- Descrição: Limpa todos os campos de entrada do formulário (TextFields e ComboBox) e remove a seleção da tabela.
- Uso: É chamado após adicionar, editar ou remover um pet, para preparar o formulário para uma nova entrada.

private boolean validarCampos()

- Descrição: Realiza uma verificação completa de todos os campos do formulário para garantir que os dados são válidos antes de serem salvos.
- Funcionamento:
  - Verifica se os campos de texto obrigatórios não estão vazios.
  - Verifica se o ComboBox do sexo foi selecionado.
  - Usa blocos try-catch para garantir que os campos de ano, peso e ID do tutor contenham números válidos.
  - Verifica se os valores numéricos estão dentro de uma faixa aceitável (ex: ano entre 1980 e o ano atual, peso e ID positivos).
- Retorno: (boolean) - Retorna true se todos os dados forem válidos; false se houver algum erro. Se false, também exibe uma janela de erro listando todos os problemas encontrados.

private void carregarPetNoFormulario(Pet pet)

- Descrição: Preenche os campos do formulário com os dados de um objeto Pet.
- Uso: É chamado pelo listener da tabela sempre que o usuário clica em uma linha, facilitando a visualização e edição dos dados.

private void mostrarErro(String mensagem) e private void mostrarSucesso(String mensagem)

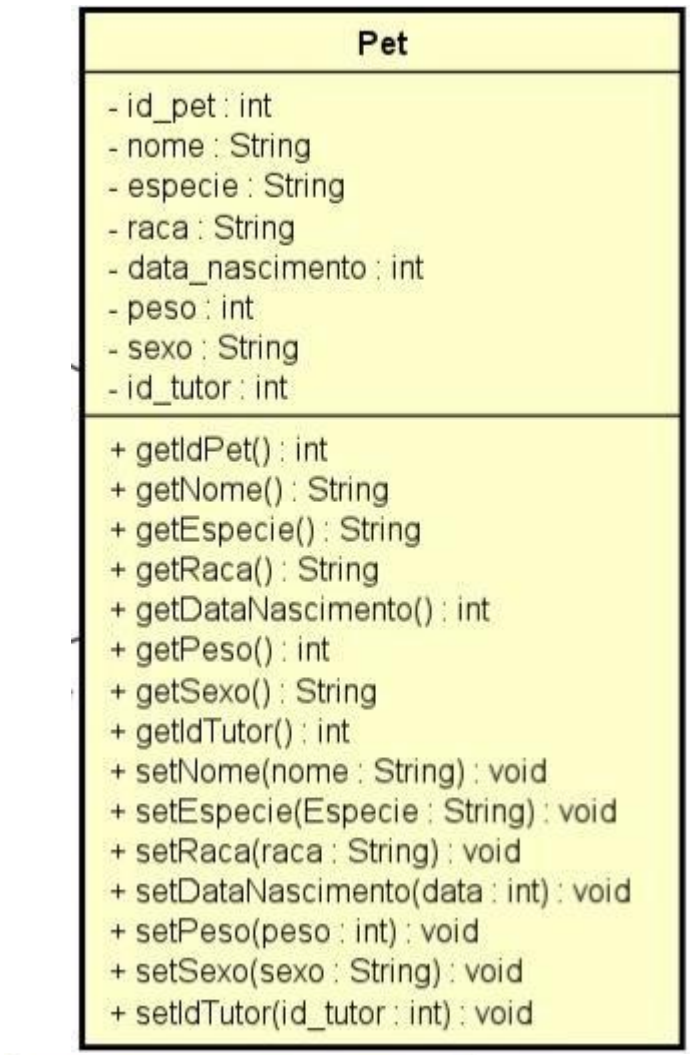
- Descrição: São métodos auxiliares que exibem caixas de diálogo (Alert) para o usuário. Um mostra uma mensagem de erro (AlertType.ERROR) e o outro uma de sucesso (AlertType.INFORMATION).
- Uso: Padronizam o feedback ao usuário e evitam a repetição de código.

private void addSampleData()

- Descrição: Adiciona dois registros de pets "de exemplo" à lista.

- Uso: É chamado na inicialização apenas se o arquivo de dados não existir e a lista estiver vazia, para que o usuário veja a aplicação já com alguns dados na tabela.

Imagem da classe correspondente



Nome: Guilherme

1. Classe Tutor

Esta classe atua como um modelo de dados (entidade) para representar um tutor de pet. Sua função é armazenar as informações do tutor e fornecer métodos para acessar e modificar esses dados.

Construtor

public Tutor(int id_tutor, String nome, String telefone, String endereco, String email)

- Descrição: Este é o construtor da classe, responsável por criar uma nova instância de Tutor e inicializar todos os seus atributos de uma só vez.
- Parâmetros:
  - id_tutor: O identificador numérico único do tutor.
  - nome: O nome completo do tutor.
  - telefone: O número de telefone de contato.
  - endereco: O endereço residencial do tutor.
  - email: O endereço de e-mail do tutor.
- Uso: É chamado no TutorController ao adicionar um novo tutor, recebendo os dados inseridos pelo usuário no formulário.

Métodos Gerais

public String toString()

- Descrição: Sobrescreve o método toString() padrão do Java para fornecer uma representação textual mais amigável do objeto Tutor.
- Retorno: (String) - Retorna o nome do tutor seguido pelo seu e-mail entre parênteses. Exemplo: "João da Silva (joao.silva@email.com)".
- Uso: Embora não seja diretamente usado na interface gráfica fornecida, é uma boa prática para depuração e para futuras funcionalidades que possam precisar exibir o tutor como uma string.

Métodos Getters e Setters (Acessores e Modificadores)

Esses métodos são essenciais para o encapsulamento, permitindo um acesso controlado aos atributos privados da classe.

public int getId_tutor()

- Descrição: Retorna o valor do atributo id_tutor.
- Retorno: (int) - O ID único do tutor.

public void setId_tutor(int id_tutor)

- Descrição: Define ou atualiza o valor do atributo id_tutor.
- Parâmetro: id_tutor - O novo ID para o tutor.

public String getNome()

- Descrição: Retorna o nome do tutor.
- Retorno: (String) - O nome completo do tutor.

public void setNome(String nome)

- Descrição: Define ou atualiza o nome do tutor.
- Parâmetro: nome - O novo nome do tutor.

public String getTelefone()

- Descrição: Retorna o número de telefone do tutor.
- Retorno: (String) - O telefone de contato.

public void setTelefone(String telefone)

- Descrição: Define ou atualiza o número de telefone do tutor.
- Parâmetro: telefone - O novo número de telefone.

public String getEndereco()

- Descrição: Retorna o endereço do tutor.
- Retorno: (String) - O endereço residencial do tutor.

public void setEndereco(String endereco)

- Descrição: Define ou atualiza o endereço do tutor.
- Parâmetro: endereco - O novo endereço do tutor.

public String getEmail()

- Descrição: Retorna o endereço de e-mail do tutor.
- Retorno: (String) - O e-mail de contato do tutor.

public void setEmail(String email)

- Descrição: Define ou atualiza o endereço de e-mail do tutor.
- Parâmetro: email - O novo e-mail do tutor.

---

2. Classe TutorController

Esta classe gerencia a interface gráfica (GUI) e a lógica de negócio para o cadastro de tutores.

Métodos do Ciclo de Vida do JavaFX

public static void main(String[] args)

- Descrição: Ponto de entrada padrão de uma aplicação Java. Ele simplesmente chama o método launch(args), que inicia o ambiente JavaFX e o ciclo de vida da aplicação.

public void start(Stage stage)

- Descrição: É o método principal do ciclo de vida da aplicação JavaFX, responsável por construir e configurar toda a interface gráfica.
- Funcionalidades:
    1. Chama carregarTutores() para carregar dados salvos anteriormente.
    2. Cria e configura os campos de texto (TextField) para entrada de dados.
    3. Cria e posiciona os botões de ação ("Adicionar", "Editar", "Remover", "Limpar").
    4. Configura a TableView e suas colunas, vinculando cada coluna a um atributo da classe Tutor através de PropertyValueFactory.
    5. Define as ações (event handlers) para cada botão, como btnAdd.setOnAction(e -> adicionarTutor()).
    6. Organiza todos os componentes visuais em layouts (VBox, HBox).
    7. Cria a cena (Scene), a insere na janela principal (Stage) e a exibe para o usuário.
- Parâmetro: stage - A janela principal da aplicação, fornecida pelo framework JavaFX.

Métodos de Ação e Lógica de Negócio

private void adicionarTutor()

- Descrição: Acionado pelo botão "Adicionar". Coleta os dados dos campos de texto, valida o ID e cria um novo tutor.
- Funcionamento:
    1. Lê os dados dos campos idField, nomeField, etc.
    2. Converte o ID para um número inteiro. Se a conversão falhar (ex: texto não numérico), captura a NumberFormatException e exibe um erro.
    3. Verifica se o ID já existe na listaTutores. Se existir, mostra um alerta e interrompe a operação.
    4. Se o ID for único, cria uma nova instância de Tutor.
    5. Adiciona o novo tutor à listaTutores (a tabela na tela é atualizada automaticamente).

6. Chama salvarTutores() para persistir a alteração e limparCampos() para resetar o formulário.

private void editarTutor()

- Descrição: Acionado pelo botão "Editar". Modifica os dados do tutor que está selecionado na tabela.
- Funcionamento:
    1. Obtém o objeto Tutor selecionado na table. Se nenhum for selecionado, exibe um alerta.
    2. Se houver um tutor selecionado, lê os novos dados dos campos de texto.
    3. Utiliza os métodos set do objeto Tutor selecionado para atualizar seus atributos.
    4. Chama table.refresh() para garantir que a tabela exiba os dados atualizados.
    5. Chama salvarTutores() e limparCampos().

private void removerTutor()

- Descrição: Acionado pelo botão "Remover". Exclui o tutor selecionado na tabela.
- Funcionamento:
    1. Obtém o objeto Tutor selecionado. Se nenhum for selecionado, exibe um alerta.
    2. Exibe uma caixa de diálogo de confirmação (Alert) para evitar exclusões acidentais.
    3. Se o usuário confirmar, o tutor é removido da listaTutores.
    4. Chama salvarTutores() e limparCampos().

Métodos Utilitários e de Interface

private void limparCampos()

- Descrição: Limpa o conteúdo de todos os campos de texto (TextField) e remove qualquer seleção na tabela.
- Uso: Chamado após operações bem-sucedidas (adicionar, editar, remover) para deixar o formulário pronto para a próxima ação.

private void preencherCampos()

- Descrição: Preenche os campos de texto do formulário com os dados do tutor selecionado na tabela.

- Uso: É acionado pelo evento setOnMouseClicked da tabela, facilitando a visualização e edição dos dados de um tutor existente.

private void mostrarAlerta(String titulo, String mensagem)

- Descrição: Método auxiliar que cria e exibe uma janela de alerta (Alert) para o usuário.
- Uso: Centraliza a lógica de exibição de mensagens de erro ou atenção, evitando código repetido.
- Parâmetros:
    - titulo: O texto que aparecerá na barra de título da janela de alerta.
    - mensagem: O conteúdo principal da mensagem a ser exibida.

Métodos de Persistência de Dados

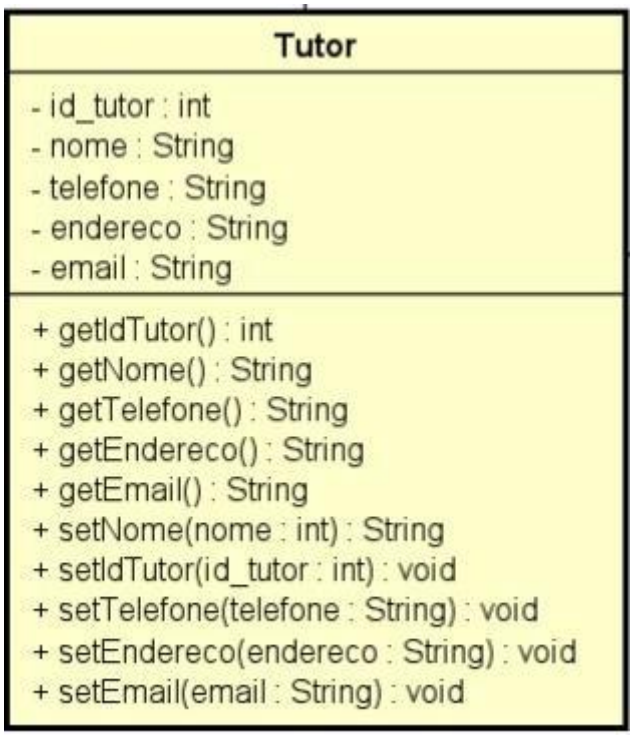
private void salvarTutores()

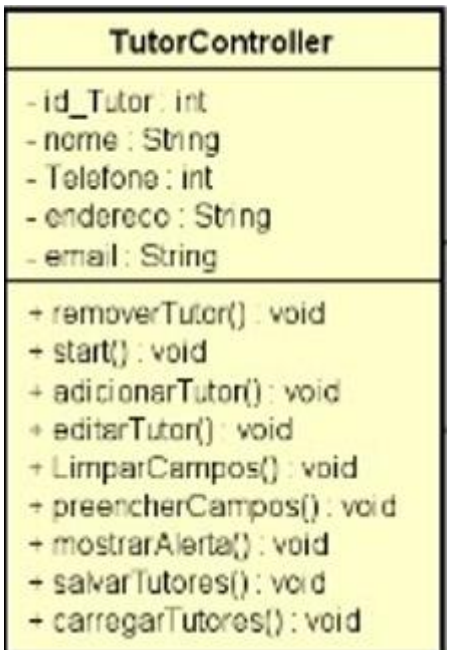
- Descrição: Salva a listaTutores atual em um arquivo binário chamado tutores.dat.
- Funcionamento: Utiliza ObjectOutputStream para serializar (converter em bytes) a lista de tutores e gravá-la no arquivo. É envolvido por um try-with-resources para garantir que o fluxo de saída seja fechado corretamente.

private void carregarTutores()

- Descrição: Carrega os dados dos tutores do arquivo tutores.dat quando a aplicação é iniciada.
- Funcionamento:
    1. Verifica se o arquivo existe.
    2. Se existir, utiliza ObjectInputStream para ler o arquivo e desserializar (reconstruir) a lista de objetos.
    3. Adiciona cada objeto lido (após verificar que é uma instância de Tutor) à listaTutores da aplicação.

Imagems da classes correspondente





Nome: Juliana

1. Classe Veterinario

Esta classe serve como um modelo de dados (entidade) para representar um veterinário. Sua função é armazenar as informações do profissional e fornecer métodos para acessar e modificar esses dados. É importante notar que no segundo bloco de código, esta classe foi definida como uma classe interna estática (static class) dentro de VeterinarioController, o que é uma forma de agrupar classes relacionadas em um único arquivo.

Construtor

public Veterinario(int id_veterinario, String nome, String crmv, String especialidade, int telefone, String email)

- Descrição: Este é o construtor da classe, responsável por criar uma nova instância de Veterinario e inicializar todos os seus atributos de uma só vez.
- Parâmetros:
    - id_veterinario: O identificador numérico único do veterinário.
    - nome: O nome completo do profissional.
    - crmv: O número de registro no Conselho Regional de Medicina Veterinária.

    - especialidade: A área de especialização do veterinário (ex: "Clínico Geral", "Cirurgião").
    - telefone: O número de telefone de contato.
    - email: O endereço de e-mail do profissional.
- Uso: É chamado no VeterinarioController ao adicionar um novo veterinário, recebendo os dados inseridos pelo usuário no formulário.

Métodos Getters e Setters (Acessores e Modificadores)

Esses métodos são essenciais para o encapsulamento, permitindo um acesso controlado aos atributos privados da classe.

public int getIdVeterinario()

- Descrição: Retorna o valor do atributo id_veterinario.
- Retorno: (int) - O ID único do veterinário.

public void setIdVeterinario(int id_veterinario)

- Descrição: Define ou atualiza o valor do atributo id_veterinario.
- Parâmetro: id_veterinario - O novo ID para o veterinário.

public String getNome()

- Descrição: Retorna o nome do veterinário.
- Retorno: (String) - O nome completo do profissional.

public void setNome(String nome)

- Descrição: Define ou atualiza o nome do veterinário.
- Parâmetro: nome - O novo nome do veterinário.

public String getCrmv()

- Descrição: Retorna o número de registro CRMV do veterinário.
- Retorno: (String) - O CRMV do profissional.

public void setCrmv(String crmv)

- Descrição: Define ou atualiza o número de registro CRMV do veterinário.
- Parâmetro: crmv - O novo número de CRMV.

public String getEspecialidade()

- Descrição: Retorna a especialidade do veterinário.
- Retorno: (String) - A área de especialização do profissional.

public void setEspecialidade(String especialidade)

- Descrição: Define ou atualiza a especialidade do veterinário.
- Parâmetro: especialidade - A nova especialidade do profissional.

public int getTelefone()

- Descrição: Retorna o número de telefone do veterinário.
- Retorno: (int) - O telefone de contato.

public void setTelefone(int telefone)

- Descrição: Define ou atualiza o número de telefone do veterinário.
- Parâmetro: telefone - O novo número de telefone.

public String getEmail()

- Descrição: Retorna o endereço de e-mail do veterinário.
- Retorno: (String) - O e-mail de contato do profissional.

public void setEmail(String email)

- Descrição: Define ou atualiza o endereço de e-mail do veterinário.
- Parâmetro: email - O novo e-mail do profissional.

---

2. Classe VeterinarioController

Esta classe gerencia a interface gráfica (GUI) e a lógica de negócio para o cadastro de veterinários.

Métodos do Ciclo de Vida do JavaFX

public static void main(String[] args)

- Descrição: Ponto de entrada padrão de uma aplicação Java. Ele simplesmente chama o método launch(args), que inicia o ambiente JavaFX e o ciclo de vida da aplicação.

public void start(Stage stage)

- Descrição: É o método principal do ciclo de vida da aplicação JavaFX, responsável por construir e configurar toda a interface gráfica.
- Funcionalidades:
    1. Chama carregarVeterinarios() para carregar dados salvos anteriormente.
    2. Cria e configura os campos de texto (TextField) para entrada de dados.
    3. Cria e posiciona os botões de ação ("Adicionar", "Editar", "Remover", "Limpar").

4. Configura a TableView e suas colunas, vinculando cada coluna a um atributo da classe Veterinario através de PropertyValueFactory.
5. Define as ações (event handlers) para cada botão, como btnAdd.setOnAction(e -> adicionarVeterinario()).
6. Organiza todos os componentes visuais em layouts (VBox, HBox).
7. Cria a cena (Scene), a insere na janela principal (Stage) e a exibe para o usuário.

- Parâmetro: stage - A janela principal da aplicação, fornecida pelo framework JavaFX.

Métodos de Ação e Lógica de Negócio

private void adicionarVeterinario()

- Descrição: Acionado pelo botão "Adicionar". Coleta os dados dos campos de texto, valida o ID e cria um novo veterinário.
- Funcionamento:
  1. Lê os dados dos campos idField, nomeField, etc.
  2. Converte o ID para um número inteiro. Se a conversão falhar (ex: texto não numérico), captura a NumberFormatException e exibe um erro.
  3. Verifica se o ID já existe na listaVeterinarios. Se existir, mostra um alerta e interrompe a operação.
  4. Se o ID for único, cria uma nova instância de Veterinario.
  5. Adiciona o novo veterinário à listaVeterinarios (a tabela na tela é atualizada automaticamente).
  6. Chama salvarVeterinarios() para persistir a alteração e limparCampos() para resetar o formulário.

private void editarVeterinario()

- Descrição: Acionado pelo botão "Editar". Modifica os dados do veterinário que está selecionado na tabela.
- Funcionamento:
  1. Obtém o objeto Veterinario selecionado na table. Se nenhum for selecionado, exibe um alerta.
  2. Se houver um veterinário selecionado, lê os novos dados dos campos de texto.

3. Utiliza os métodos set do objeto Veterinario selecionado para atualizar seus atributos.
4. Chama table.refresh() para garantir que a tabela exiba os dados atualizados.
5. Chama salvarVeterinarios() e limparCampos().

private void removerVeterinario()

- Descrição: Acionado pelo botão "Remover". Exclui o veterinário selecionado na tabela.
- Funcionamento:
    1. Obtém o objeto Veterinario selecionado. Se nenhum for selecionado, exibe um alerta.
    2. Exibe uma caixa de diálogo de confirmação (Alert) para evitar exclusões acidentais.
    3. Se o usuário confirmar, o veterinário é removido da listaVeterinarios.
    4. Chama salvarVeterinarios() e limparCampos().

Métodos Utilitários e de Interface

private void limparCampos()

- Descrição: Limpa o conteúdo de todos os campos de texto (TextField) e remove qualquer seleção na tabela.
- Uso: Chamado após operações bem-sucedidas (adicionar, editar, remover) para deixar o formulário pronto para a próxima ação.

private void preencherCampos()

- Descrição: Preenche os campos de texto do formulário com os dados do veterinário selecionado na tabela.
- Uso: É acionado pelo evento setOnMouseClicked da tabela, facilitando a visualização e edição dos dados de um veterinário existente.

private void mostrarAlerta(String titulo, String mensagem)

- Descrição: Método auxiliar que cria e exibe uma janela de alerta (Alert) para o usuário.
- Uso: Centraliza la lógica de exibição de mensagens de erro ou atenção, evitando código repetido.
- Parâmetros:
    - titulo: O texto que aparecerá na barra de título da janela de alerta.

- mensagem: O conteúdo principal da mensagem a ser exibida.

Métodos de Persistência de Dados

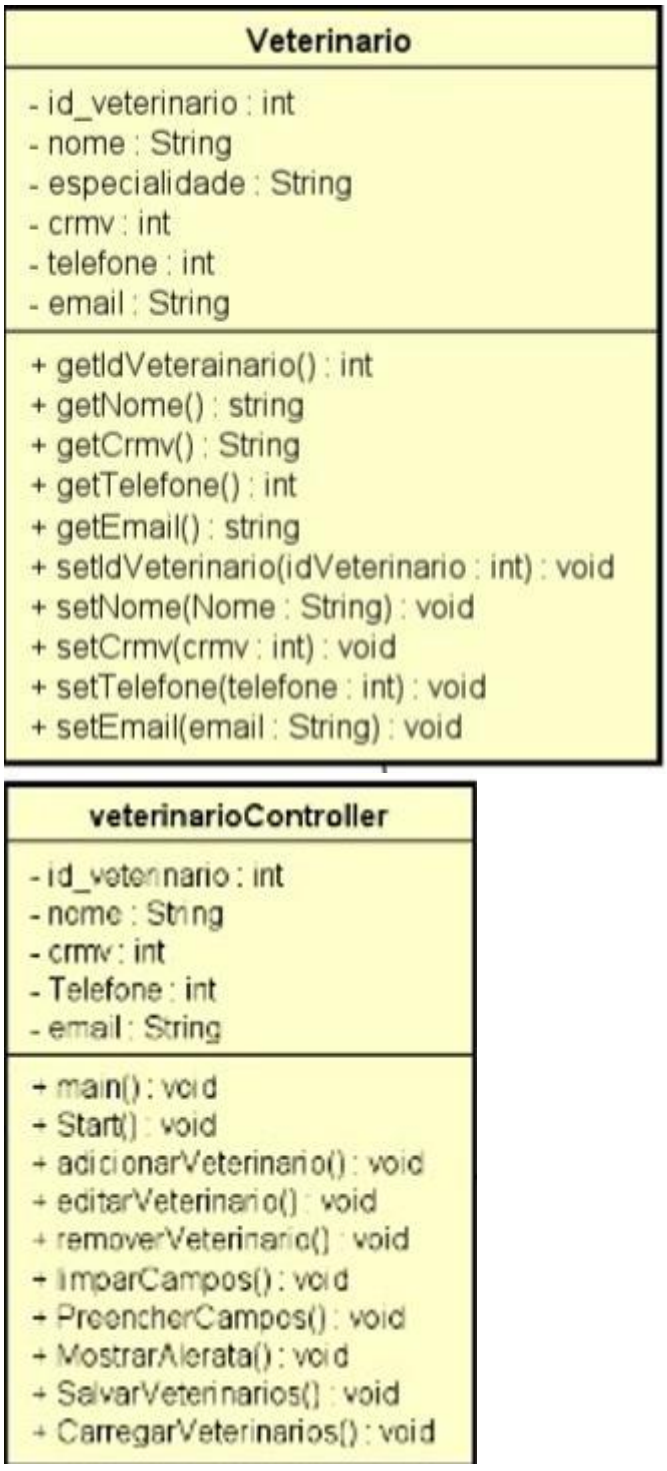
private void salvarVeterinarios()

- Descrição: Salva a listaVeterinarios atual em um arquivo binário chamado veterinarios.dat.
- Funcionamento: Utiliza ObjectOutputStream para serializar (converter em bytes) a lista de veterinários e gravá-la no arquivo. É envolvido por um try-with-resources para garantir que o fluxo de saída seja fechado corretamente.

private void carregarVeterinarios()

- Descrição: Carrega os dados dos veterinários do arquivo veterinarios.dat quando a aplicação é iniciada.
- Funcionamento:
    1. Verifica se o arquivo existe.
    2. Se existir, utiliza ObjectInputStream para ler o arquivo e desserializar (reconstruir) a lista de objetos.
    3. Adiciona cada objeto lido (após verificar que é uma instância de Veterinario) à listaVeterinarios da aplicação.

Imagens das classes correspondente



Nome: Cibely

1. Classe Consulta

Esta classe é um POJO (Plain Old Java Object) que representa a entidade "Consulta". Sua única responsabilidade é armazenar os dados de uma consulta veterinária. Ela implementa Serializable para que seus objetos possam ser convertidos em bytes e salvos em arquivo.

Construtores

public Consulta()

- Descrição: Construtor padrão sem argumentos. Permite criar uma

instância vazia da classe.

- Uso: Útil para frameworks ou para criar um objeto que será preenchido posteriormente através dos métodos setters.

public Consulta(int id_consulta, int id_pet, ...)

- Descrição: Construtor parametrizado que inicializa todos os atributos do objeto no momento de sua criação.
- Parâmetros:
    - id_consulta: O ID único da consulta.
    - id_pet: O ID do pet associado à consulta.

- id_veterinario: O ID do veterinário que realizou a consulta.
- data: A data e hora da consulta, como um objeto LocalDateTime.
- hora: A hora da consulta, como uma String no formato "HH:mm".

Métodos Getters e Setters (Acessores e Modificadores)

Estes métodos fornecem acesso controlado aos atributos privados da classe, seguindo o princípio do encapsulamento.

- public int getId_consulta(): Retorna o ID da consulta.
- public void setId_consulta(int id_consulta): Define o ID da consulta.
- public int getId_pet(): Retorna o ID do pet.
- public void setId_pet(int id_pet): Define o ID do pet.
- public int getId_veterinario(): Retorna o ID do veterinário.
- public void setId_veterinario(int id_veterinario): Define o ID do veterinário.
- public LocalDateTime getData(): Retorna a data e hora da consulta.
- public void setData(LocalDateTime data): Define a data e hora da consulta.
- public String getHora(): Retorna a hora da consulta (string).
- public void setHora(String hora): Define a hora da consulta (string).

Métodos Adicionais

public String toString()

- Descrição: Sobrescreve o método padrão toString() para fornecer uma representação textual formatada e legível do objeto Consulta, útil para depuração e logs.
- Retorno: Uma String formatada com os principais dados da consulta.

---

2. Classe ConsultaDAO

Esta classe implementa o padrão DAO, que isola a lógica de acesso a dados do resto da aplicação. Sua responsabilidade é exclusivamente ler e escrever os dados das consultas no arquivo consultas.dat.

public void salvarConsultas(List<Consulta> consultas)

- Descrição: Serializa e salva a lista completa de consultas no arquivo.
- Funcionamento: Utiliza um ObjectOutputStream para escrever a lista de objetos no arquivo. O uso de try-with-resources garante que o stream de arquivo seja fechado corretamente, mesmo em caso de erro.

- Parâmetro: consultas - A lista de objetos Consulta a ser salva.

public List<Consulta> carregarConsultas()

- Descrição: Lê e desserializa a lista de consultas do arquivo.
- Funcionamento: Se o arquivo não existir, retorna uma nova lista vazia. Caso contrário, usa um ObjectInputStream para ler os objetos do arquivo e os converte de volta para uma List<Consulta>.
- Retorno: A List<Consulta> carregada do arquivo ou uma lista vazia.

public void inserir(Consulta consulta)

- Descrição: Adiciona uma nova consulta ao arquivo.
- Funcionamento: Carrega a lista atual, adiciona a nova consulta à lista em memória e, em seguida, salva a lista inteira de volta no arquivo.
- Parâmetro: consulta - O novo objeto Consulta a ser inserido.

public boolean atualizar(Consulta consultaAtualizada)

- Descrição: Atualiza uma consulta existente no arquivo.
- Funcionamento: Carrega a lista, procura pela consulta com o mesmo id_consulta, a substitui pela consultaAtualizada e salva a lista de volta.
- Retorno: true se a consulta foi encontrada e atualizada; false caso contrário.

public boolean excluir(int id)

- Descrição: Remove uma consulta do arquivo com base no seu ID.
- Funcionamento: Carrega a lista, usa o método removeIf com uma expressão lambda para encontrar e remover a consulta correspondente, e então salva a lista modificada.
- Retorno: true se a consulta foi encontrada e removida; false caso contrário.

public Consulta buscarPorId(int id)

- Descrição: Procura e retorna uma única consulta pelo seu ID.
- Funcionamento: Carrega a lista e utiliza a API de Streams do Java (.stream().filter().findFirst()) para encontrar a consulta de forma eficiente.
- Retorno: O objeto Consulta encontrado ou null se não existir.

public List<Consulta> listarTodas()

- Descrição: Retorna a lista completa de todas as consultas salvas.
- Funcionamento: É um método de conveniência que simplesmente chama carregarConsultas().

---

3. Classe ConsultaCRUDApp

Esta classe gerencia toda a interface gráfica (Visão) e a lógica de interação com o usuário (Controle).

Métodos do Ciclo de Vida e de Construção da UI

public void start(Stage primaryStage)

- Descrição: Ponto de entrada da aplicação JavaFX. Constrói a janela e organiza todos os componentes visuais.
- Funcionamento:
    1. Cria o layout principal (BorderPane).
    2. Chama criarFormulario() e criarAreaTabela() para construir as seções da UI.
    3. Chama configurarEventos() para definir as ações dos botões.
    4. Chama atualizarTabela() para carregar os dados iniciais.
    5. Monta e exibe a cena (Scene).

private VBox criarFormulario()

- Descrição: Cria e retorna o painel do formulário com todos os campos de entrada (TextFields, DatePicker) e botões.
- Retorno: Um VBox contendo o formulário de cadastro.

private VBox criarAreaTabela()

- Descrição: Cria e retorna o painel que contém a tabela de consultas.
- Funcionamento:
    - Configura as colunas da TableView.
    - Usa PropertyValueFactory para vincular colunas aos getters da classe Consulta.
    - Para a coluna de data, usa um setCellValueFactory customizado para formatar o LocalDateTime para o padrão "dd/MM/yyyy".
    - Adiciona um "ouvinte" (listener) que chama preencherFormulario sempre que uma linha da tabela é selecionada.
- Retorno: Um VBox contendo a tabela e seus componentes associados.

private void configurarEventos()

- Descrição: Centraliza a atribuição das ações a cada botão, ligando o clique de um botão a um método específico (ex: btnInserir chama inserirConsulta).

Métodos de Lógica de Aplicação (Event Handlers)

private void inserirConsulta()

- Descrição: Lida com a lógica de inserção de uma nova consulta.
- Funcionamento:
    1. Chama criarConsultaDoFormulario() para obter os dados da UI.
    2. Gera um novo ID único para a consulta, buscando o maior ID existente e incrementando-o.
    3. Chama consultaDAO.inserir() para persistir os dados.
    4. Exibe uma mensagem de sucesso e atualiza a UI chamando limparFormulario() e atualizarTabela().

private void atualizarConsulta()

- Descrição: Lida com a lógica de atualização de uma consulta existente.
- Funcionamento:
    1. Verifica se uma consulta foi selecionada (se o campo ID não está vazio).
    2. Obtém os dados do formulário e o ID da consulta a ser atualizada.
    3. Chama consultaDAO.atualizar().
    4. Atualiza a UI e informa o usuário sobre o sucesso ou falha da operação.

private void excluirConsulta()

- Descrição: Lida com a lógica de exclusão de uma consulta.
- Funcionamento:
    1. Verifica se uma consulta foi selecionada.
    2. Exibe um Alert de confirmação para evitar exclusão acidental.
    3. Se confirmado, chama consultaDAO.excluir() com o ID da consulta.
    4. Atualiza a UI e informa o usuário.

Métodos Utilitários

private void atualizarTabela()

- Descrição: Recarrega todos os dados do arquivo através do DAO e os exibe na tabela. É o método central para manter a UI sincronizada com os dados.

private void limparFormulario()

- Descrição: Limpa todos os campos de entrada do formulário e deseleciona qualquer linha na tabela.

private void preencherFormulario(Consulta consulta)

- Descrição: Preenche os campos do formulário com os dados de uma consulta selecionada na tabela, facilitando a edição.

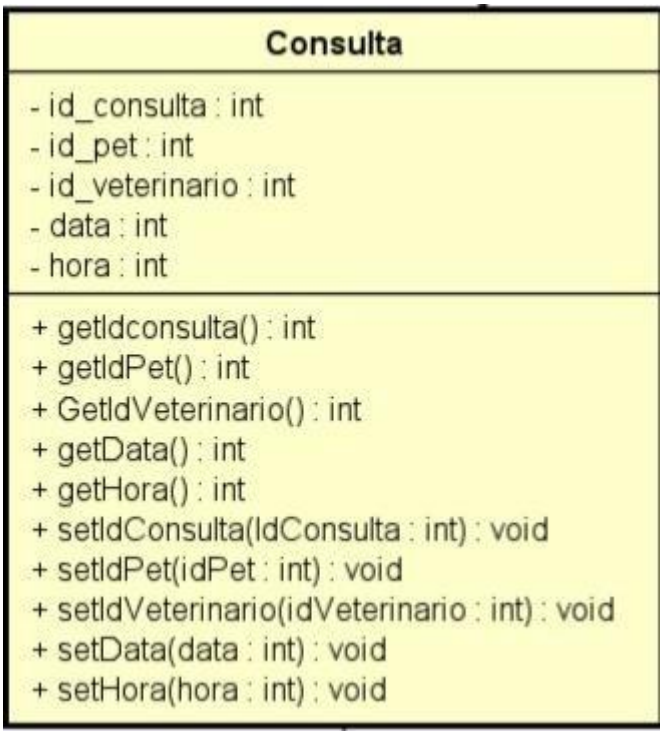
private Consulta criarConsultaDoFormulario()

- Descrição: Método auxiliar que lê os dados de todos os campos do formulário e cria um novo objeto Consulta a partir deles. Ajuda a evitar duplicação de código.
- Retorno: Um objeto Consulta com os dados da UI.

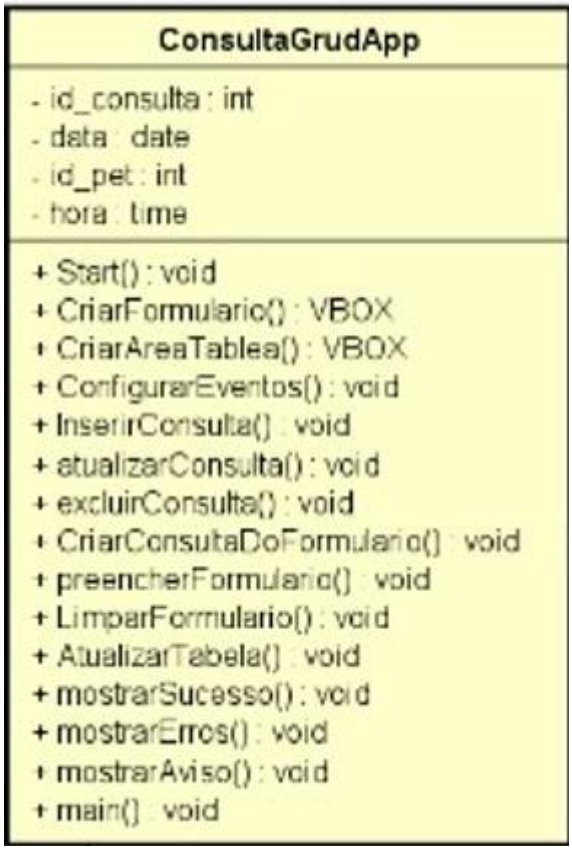
private void mostrarSucesso(String msg), mostrarErro(String msg), mostrarAviso(String msg)

- Descrição: Métodos auxiliares que exibem diferentes tipos de Alert (sucesso, erro, aviso) para fornecer feedback consistente ao usuário.

Imagems das classes correspondente




ConsultaDAO
- Arquivo : String
- separador : String
- formatoData : date
+ ConsultaParaTexto() : String
+ TextoParaConsulta() : Consulta
+ SalvarConsultas() : void
+ CarregarConsulta() : void
+ inserir() : void
+ atualizar() : boolean
+ excluir() : boolean


ConsultaGrudApp
- id_consulta : int
- data : date
- id_pet : int
- hora : time
+ Start() : void
+ CriarFormulario() : VBOX
+ CriarAreaTabela() : VBOX
+ ConfigurarEventos() : void
+ InserirConsulta() : void
+ atualizarConsulta() : void
+ excluirConsulta() : void
+ CriarConsultaDoFormulario() : void
+ preencherFormulario() : void
+ LimparFormulario() : void
+ AtualizarTabela() : void
+ mostrarSucesso() : void
+ mostrarErros() : void
+ mostrarAviso() : void
+ main() : void

Nome: Angely

1. Classe Diagnostico

Esta classe é um modelo de dados (entidade ou POJO) que representa um diagnóstico veterinário. Sua principal responsabilidade é encapsular as informações de um diagnóstico e fornecer métodos para acessar e modificar esses dados. A implementação da interface Serializable é crucial, pois permite que objetos desta classe sejam convertidos em bytes para serem salvos em arquivo.

Construtores

public Diagnostico(int id_diagnostico, int id_consulta, ...)

- Descrição: Este é o construtor completo da classe. Ele é utilizado para criar uma instância de Diagnostico com todos os atributos já definidos, incluindo o id_diagnostico.
- Uso Típico: Ideal para reconstruir um objeto a partir de dados carregados de um arquivo ou banco de dados, onde o ID já existe.

public Diagnostico(int id_consulta, String nomeDiagnostico, ...)

- Descrição: Este é um construtor de conveniência para criar um novo diagnóstico. Ele chama o construtor completo usando a palavra-chave this(), passando 0 como um valor temporário para o id_diagnostico. O ID real será gerado e atribuído pelo DiagnosticoController.
- Uso Típico: Utilizado ao criar um diagnóstico pela primeira vez através da interface do usuário.

Métodos Getters (Acessores)

Estes métodos permitem a leitura dos valores dos atributos privados da classe.

- public int getId_diagnostico(): Retorna o ID único do diagnóstico.
- public int getId_consulta(): Retorna o ID da consulta à qual este diagnóstico está associado.
- public String getNomeDiagnostico(): Retorna o nome ou a descrição do diagnóstico (ex: "Gripe Canina").
- public LocalDate getDataDiagnostico(): Retorna a data em que o diagnóstico foi feito, como um objeto LocalDate.
- public String getTratamento(): Retorna a descrição do tratamento recomendado.
- public String getMedicamentosPrescritos(): Retorna a lista de medicamentos que foram prescritos.

Métodos Setters (Modificadores)

Estes métodos permitem a alteração dos valores dos atributos privados. São essenciais para a funcionalidade de "Editar" no controller.

- public void setId_diagnostico(int id_diagnostico): Define ou atualiza o ID do diagnóstico.
- public void setId_consulta(int id_consulta): Define ou atualiza o ID da consulta associada.
- public void setNomeDiagnostico(String nomeDiagnostico): Define ou atualiza o nome do diagnóstico.
- public void setDataDiagnostico(LocalDate dataDiagnostico): Define ou atualiza a data do diagnóstico.
- public void setTratamento(String tratamento): Define ou atualiza a descrição do tratamento.
- public void setMedicamentosPrescritos(String medicamentosPrescritos): Define ou atualiza os medicamentos prescritos.

Métodos Adicionais

public String toString()

- Descrição: Sobrescreve o método padrão toString() para fornecer uma representação textual clara e legível do objeto Diagnostico, útil para fins de depuração e logs.
- Retorno: Uma String formatada contendo todos os atributos do diagnóstico.

---

2. Classe DiagnosticoController

Esta classe é o coração da aplicação, responsável por gerenciar a interface gráfica (GUI), a lógica de negócio e a persistência dos dados de diagnósticos.

Métodos do Ciclo de Vida do JavaFX

public static void main(String[] args)

- Descrição: Ponto de entrada padrão para uma aplicação Java. Sua única função é chamar launch(args), que inicializa o toolkit JavaFX e o ciclo de vida da aplicação.

public void start(Stage stage)

- Descrição: É o método principal do JavaFX, onde toda a interface do usuário é construída e configurada.
- Funcionalidades Detalhadas:
    1. Carregamento Inicial: Chama carregarDiagnosticos() para popular a listaDiagnosticos com dados de sessões anteriores.

2. Criação dos Componentes: Inicializa todos os componentes da UI (TextField, DatePicker, TextArea, botões, etc.).
3. Layout do Formulário: Organiza os componentes de entrada em um VBox para criar o formulário de cadastro.
4. Configuração da Tabela (TableView):
    - Cria cada TableColumn.
    - Usa PropertyValueFactory para vincular cada coluna ao getter correspondente na classe Diagnostico (ex: new PropertyValueFactory<>("nomeDiagnostico") chama getNomeDiagnostico()).
    - Formatação de Data: Para a coluna de data, utiliza setCellFactory para customizar como a data (LocalDate) é exibida, formatando-a para o padrão "dd/MM/yyyy", tornando-a mais legível para o usuário.
5. Vinculação de Eventos:
    - Associa os métodos de ação (adicionarDiagnostico, editarDiagnostico, etc.) aos eventos onAction dos botões.
    - Associa o método preencherCampos() ao evento onMouseClicked da tabela, para que um clique em uma linha preencha o formulário.
6. Montagem Final: Organiza o formulário e a tabela em um HBox principal, cria a Scene, define o título da janela e a exibe.

Métodos de Ação e Lógica de Negócio

private void adicionarDiagnostico()

- Descrição: Acionado pelo botão "Adicionar". Valida os dados do formulário e cria um novo diagnóstico.
- Funcionamento:
    1. Chama validarCampos(). Se a validação falhar, a execução é interrompida.
    2. Lê e converte os dados dos campos. Um bloco try-catch trata possíveis NumberFormatException no campo "ID da Consulta".
    3. Gera um novo ID único usando a variável nextId e o incrementa.
    4. Cria uma nova instância de Diagnostico.
    5. Adiciona o novo objeto à listaDiagnosticos, o que atualiza a tabela automaticamente.
    6. Chama salvarDiagnosticos() e limparCampos().

private void editarDiagnostico()

- Descrição: Acionado pelo botão "Editar". Atualiza os dados de um diagnóstico selecionado.
- Funcionamento:
    1. Obtém o Diagnostico selecionado na tabela. Se nada estiver selecionado, exibe um alerta.
    2. Valida os novos dados no formulário através de validarCampos().
    3. Usa os métodos set do objeto selecionado para atualizar seus valores com os dados do formulário.
    4. Chama table.refresh(). Este passo é crucial para forçar a TableView a redesenhar a linha alterada, já que a classe Diagnostico não usa JavaFX Properties.
    5. Chama salvarDiagnosticos() e limparCampos().

private void removerDiagnostico()

- Descrição: Acionado pelo botão "Remover". Exclui o diagnóstico selecionado.
- Funcionamento:
    1. Obtém o Diagnostico selecionado.
    2. Exibe um Alert de confirmação para evitar a remoção acidental.
    3. Se o usuário confirmar, remove o objeto da listaDiagnosticos e chama salvarDiagnosticos().

Métodos Utilitários e de Interface

private void preencherCampos()

- Descrição: Popula os campos do formulário com os dados do item selecionado na tabela, facilitando a visualização e edição.

private void limparCampos()

- Descrição: Limpa todos os campos de entrada do formulário e remove a seleção da tabela, preparando a UI para uma nova operação.

private boolean validarCampos()

- Descrição: Realiza uma verificação completa dos campos do formulário.
- Funcionamento: Utiliza um StringBuilder para acumular todas as mensagens de erro. Verifica campos obrigatórios, se o ID da consulta é um número válido e positivo, etc. Se houver erros, exibe um único Alert com todos os problemas.
- Retorno: true se todos os campos são válidos, false caso contrário.

private void mostrarAlerta(String titulo, String mensagem)

- Descrição: Método auxiliar reutilizável para exibir janelas de alerta (Alert) informativas ao usuário.

Métodos de Persistência de Dados

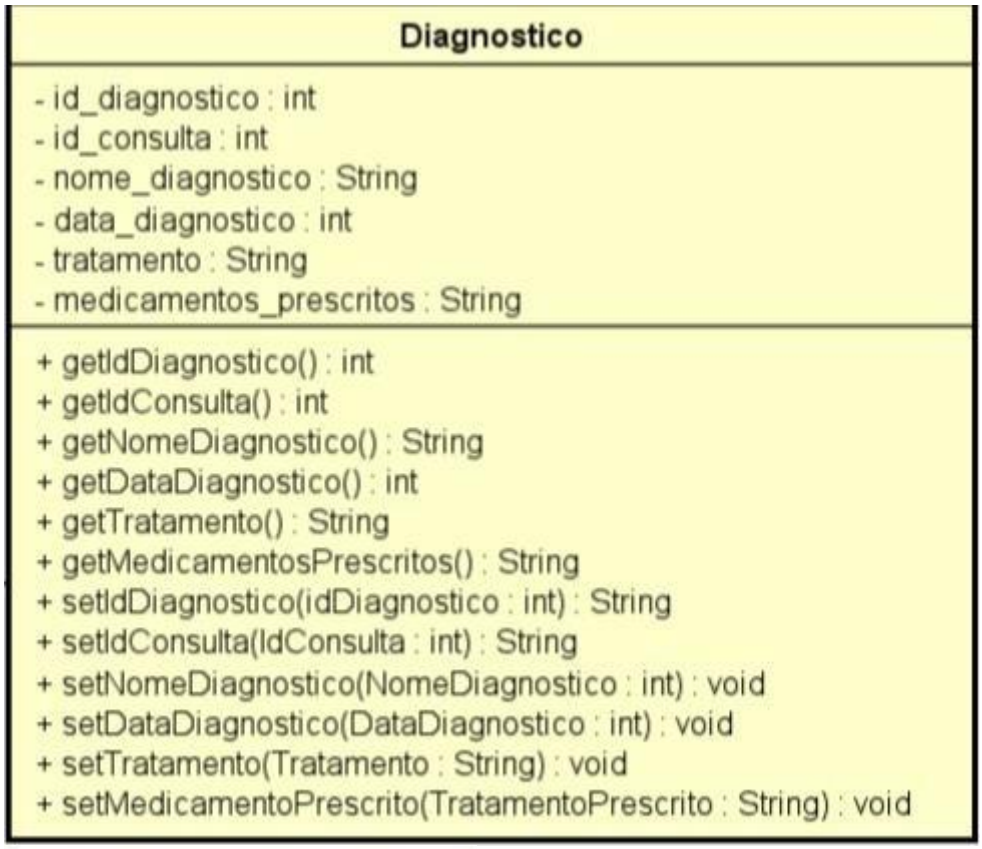
private void salvarDiagnosticos()

- Descrição: Salva a listaDiagnosticos no arquivo diagnosticos.dat.
- Funcionamento: Utiliza ObjectOutputStream para serializar a lista (convertida para um ArrayList) e gravá-la em disco.

private void carregarDiagnosticos()

- Descrição: Carrega os diagnósticos do arquivo diagnosticos.dat ao iniciar a aplicação.
- Funcionamento:
    1. Verifica se o arquivo existe e não está vazio.
    2. Usa ObjectInputStream para ler o arquivo e desserializar os dados de volta para um ArrayList<Diagnostico>.
    3. Popula a listaDiagnosticos com os dados carregados.
    4. Atualiza a variável nextId para o maior ID encontrado + 1, garantindo a continuidade da geração de IDs únicos.
    5. Trata exceções comuns como EOFException (arquivo vazio) e outras exceções de I/O

Imagens das classes correspondente



Classe MainApp

- Esta classe representa a tela inicial da aplicação de gerenciamento de uma clínica veterinária.

- Ela é responsável por exibir uma interface gráfica com botões que direcionam o usuário para diferentes cadastros e funcionalidades do sistema.

- Construtores
- java

- Copiar
- Editar

public MainApp()
Descrição:
Construtor padrão da classe MainApp.
Como a classe herda de Application, o construtor normalmente não é utilizado diretamente, pois o JavaFX inicializa a aplicação chamando o método start.

Métodos
public void start(Stage primaryStage)
Descrição:
Inicializa a interface gráfica principal da aplicação.

Parâmetros:

primaryStage: O palco principal onde os componentes gráficos da aplicação serão exibidos.

Funcionamento:
O método cria cinco botões principais, cada um com sua respectiva função:

Cadastro de Pet:
Abre uma nova janela que chama o PetController.

Cadastro de Tutor:
Abre uma nova janela que chama o TutorController.

Cadastro de Veterinário:
Abre uma nova janela que chama o VeterinarioController.

Criar Diagnóstico:
Abre uma nova janela que chama o DiagnosticoController.

Criar Consulta:
Abre uma nova janela que chama o ConsultaCRUDApp.

Uso:
Este método é chamado automaticamente ao executar a aplicação com o método launch(args).

public static void main(String[] args)
Descrição:
Ponto de entrada da aplicação JavaFX.

Parâmetros:

args: Argumentos da linha de comando (não utilizados nesta aplicação).

Uso:
Responsável por iniciar a aplicação JavaFX chamando o método launch(args).

Estrutura visual criada:
A tela inicial contém um layout vertical (VBox) com espaçamento entre os botões.

Estilo visual básico com padding e alinhamento central.

Tamanho da janela: 300x300 pixels.

Exemplo de Interface gerada:
Título da janela: Sistema de Gerenciamento

Botões exibidos na ordem:

Cadastro de Pet

Cadastro de Tutor

Cadastro de Veterinário

Criar Diagnóstico

Criar Consulta

Imagen da classe

Conclusões finais.

Nome: Lucas

No desenvolvimento do módulo de gerenciamento de pets, foi criado a classe Pet para modelar os dados e a PetController para gerenciar a interface gráfica em JavaFX e a lógica do sistema. A interface foi construída programaticamente e os dados foram salvos em arquivo usando a serialização de objetos.

Os maiores desafios foram sincronizar o formulário com a tabela de dados, gerenciar a persistência em arquivo (incluindo a geração de IDs únicos e o tratamento de erros) e implementar uma validação de entrada robusta para o usuário.

Nome: Juliana Schreiner Xavier

Desenvolver o módulo de veterinários para o projetor , foi um processo desafiador, mas muito recompensador. A principal dificuldade foi a construção de toda a interface visual (a tela) diretamente no código, sem usar ferramentas como o FXML ou o Scene Builder. Isso significou que cada botão, campo de texto e a própria tabela precisaram ser criados e organizados linha por linha, tornando o código mais longo e exigindo uma atenção extra à organização.

Apesar disso, a experiência trouxe aprendizados importantes. A forma como os dados dos veterinários foram estruturados facilitou bastante as atualizações da tela, o que simplificou a lógica interna do sistema.

No final, o desenvolvimento foi um exercício prático em como organizar um software. Conseguimos um módulo funcional que gerencia os veterinários, e essa experiência nos deu uma base sólida para entender como construir e manter aplicações complexas no futuro.

Nome: Guilherme Silva Krast

Para desenvolver o módulo de criação de tutores foi criado duas classes uma classe Tutor e outra classe TutorController. Na classe Tutor é onde se encontra os atributos e o construtor da classe junto com os geters e seters para que seja possível a criação de novos tutores, na classe TutorController é onde se encontra todo o gerenciamento da interface gráfica e onde podemos criar, salvar e editar a lista de tutores que foi criada.

Nesse projeto foi um desafio criar a interface gráfica usando apenas o JavaFX sem usar nenhum tipo de Scene builder e a persistência de objetos que salva os dados criados pelos usuários. Uma ferramenta que foi ensinada na matéria e que se tornou muito útil para manter o funcionamento do sistema sem dar erros foi o try e catch.

Certamente irei levar esses conhecimentos adquiridos para futuros projetos pois me ajudaram a aperfeiçoar os meus códigos.

Nome: Angely zambrano

O desenvolvimento das classes Diagnostico e DiagnosticoController para o nosso sistema de clínica de pets foi uma jornada de aprendizado e aplicação prática de conceitos importantes de programação. A escolha de ter uma classe de modelo separada (Diagnostico) e um controlador dedicado (DiagnosticoController) reflete uma boa prática, mesmo que a interface gráfica tenha sido construída diretamente no código Java, sem FXML.

A classe Diagnostico serviu como o modelo de dados essencial, definindo a estrutura das informações. Sua implementação Serializable foi crucial, permitindo a persistência dos dados em arquivos, garantindo que as informações não se perdessem ao fechar a aplicação. Os construtores, *getters* e *setters* foram padrões, mas fundamentais para o encapsulamento e a manipulação segura dos dados.

Já a classe DiagnosticoController atuou como o **orquestrador principal**. Foi aqui que toda a interface de usuário foi construída diretamente no código Java, e onde a lógica de negócios para adicionar, editar e remover diagnósticos foi implementada. A validação de campos e os métodos de salvamento/carregamento de dados foram vitais para a integridade e persistência do sistema.

Nome: Cibely Martins

Implementação da interface com JavaFX: Como toda a interface foi construída programaticamente, sem o uso de ferramentas visuais como FXML ou Scene Builder, foi necessário ter atenção redobrada à estrutura e à organização do código. Posicionar manualmente cada componente da interface (botões, campos, tabelas) exigiu paciência e domínio da hierarquia de layouts.

Organização do projeto: Gerenciar as classes, especialmente mantendo a separação entre modelo, controle e visão, foi desafiador. Houve momentos em que a estrutura do projeto ficou confusa, o que exigiu reestruturações para manter a clareza e a manutenibilidade do sistema.

Persistência dos dados: Utilizar a serialização de objetos para salvar e carregar informações em arquivos exigiu cuidado, especialmente na manipulação de exceções e na manutenção da integridade dos dados salvos.

Referencias

2022-06-, P. **JavaFX documentation project**. Disponível em: <https://fxdocs.github.io/docs/html5/>. Acesso em: 13 jun. 2025.

Java platform, standard edition (java SE) 8. Disponível em: <https://docs.oracle.com/javase/8/javase-clienttechnologies.htm>. Acesso em: 13 jun. 2025.

JENKOV, J. **JavaFX tutorial**. Disponível em: <https://jenkov.com/tutorials/javafx/index.html>. Acesso em: 13 jun. 2025.

DEITEL, P. J.; DEITEL, H. M. Java: Como Programar. 2016.

Object streams. Disponível em: <https://docs.oracle.com/javase/tutorial/essential/io/objectstreams.html>. Acesso em: 13 jun. 2025.