Faculdade de Informática e Administração Paulista

Giovanna Revito Roz - RM558981

Kaian Gustavo de Oliveira Nascimento - RM558986

Lucas Kenji Kikuchi - RM554424

Domain Driven Design

PortoAutoTech

Sprint 3

INTEGRANTES

RM (SOMENTE NÚMEROS)	NOME COMPLEMENTO (SEM ABREVIAR)
558981	Giovanna Revito Roz
558986	Kaian Gustavo de Oliveira Nascimento
554424	Lucas Kenji Kikuchi

SUMÁRIO

1 – Descrição

A plataforma PortoAutoTech visa a implementação de uma interface para auxiliar clientes decorrentes e novos clientes da Porto Seguro, especificamente aqueles que possuem pouco ou nenhum conhecimento sobre problemas relacionados a carros. Um sistema tecnológico e inovador, que resolve de maneira mais fácil. Com mais precisão. Mais rapidez. Mais facilidade. O objetivo é que, até o fim do projeto, o sistema incorpore uma interface que permita ao cliente ter:

- um autodiagnóstico do problema apresentado pelo veículo através de uma I.A treinada através do Machine Learning em Python, incorporando no Chatbot;
- a possibilidade de se cadastrar com suas informações pessoais (Nome, telefone, e-mail etc.) e informações de seu veículo (modelo, marca, ano…) através de nosso aplicativo / website;
- a possibilidade de agendar um serviço através do Chatbot, informando o tipo de serviço, a oficina e a data;
- um pré orçamento, com um cálculo baseado no valor do serviço + valor das peças, e a estimativa de prazo de término do serviço definido com antecedência;
- a localização de oficinas mais próximas, informando a disponibilidade de peças e agendamento de determinados serviços com a integração com um banco de dados da oficina;
- mapeamento do carro, informando através de uma notificação automática quando uma manutenção preventiva deve ser feita, baseado na última vez que o cliente realizou um serviço no carro;
- notificação automática da disponibilidade de uma determinada peça, que será acionada quando a peça for registrada como disponível no banco de dados.
- interface que informa os pontos mais próximos de carregamento para carros elétricos;
- progresso da manutenção, visualizada através de uma barra de progresso, indicando cada mudança importante sobre o status da manutenção, conforme o serviço é feito (manualmente alterado pelo mecânico);
- reconhecimento de imagem através da I.A, que será responsável por analisar a imagem do problema enviado pelo usuário, retornando à solução mais plausível;

- comunicação através de voz (speech-to-text) com o Chatbot, permitindo descrever o problema oralmente ou enviar ruídos emitidos pelo carro, que serão analisados pela I.A;
- ligação em chamada com mecânico através do Chatbot, explicando o progresso da manutenção.

Através das funcionalidades descritas, o sistema será capaz de auxiliar pessoas que enfrentam dificuldades no entendimento dos problemas do veículo, além de fornecer diferenciais que auxiliarão em necessidades dos clientes que poucas empresas oferecem.

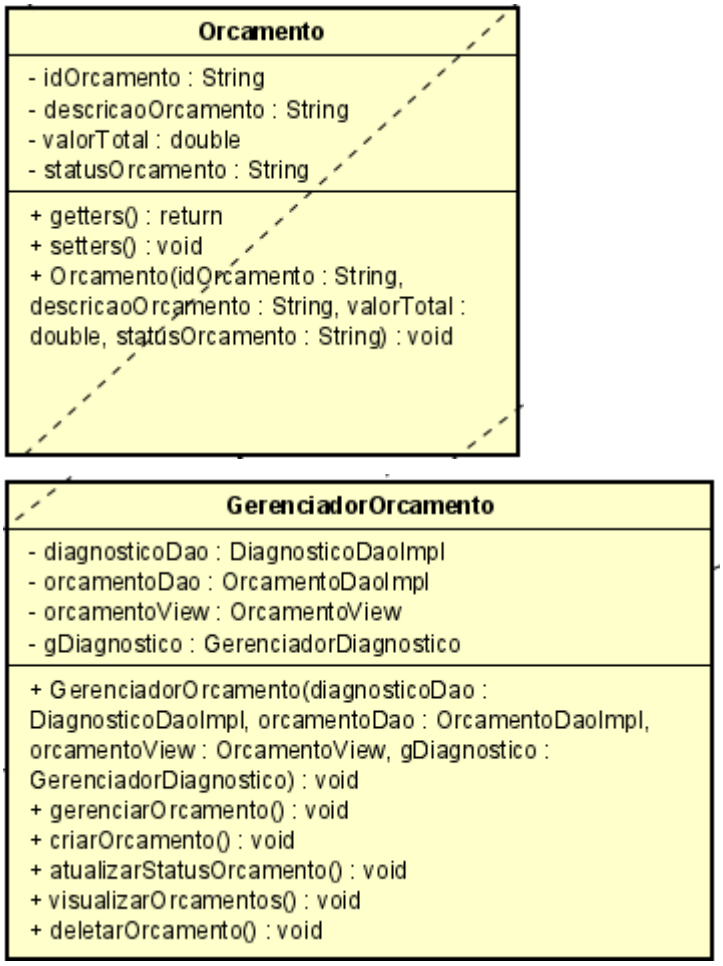
O sistema em Java, no caso, será responsável pela integração com o banco de dados, recebendo informações como dados de usuários, veículos, peças, agendamentos, diagnósticos e orçamentos.

2 – Introdução e Implementação com MVC (Model-View-Controller)

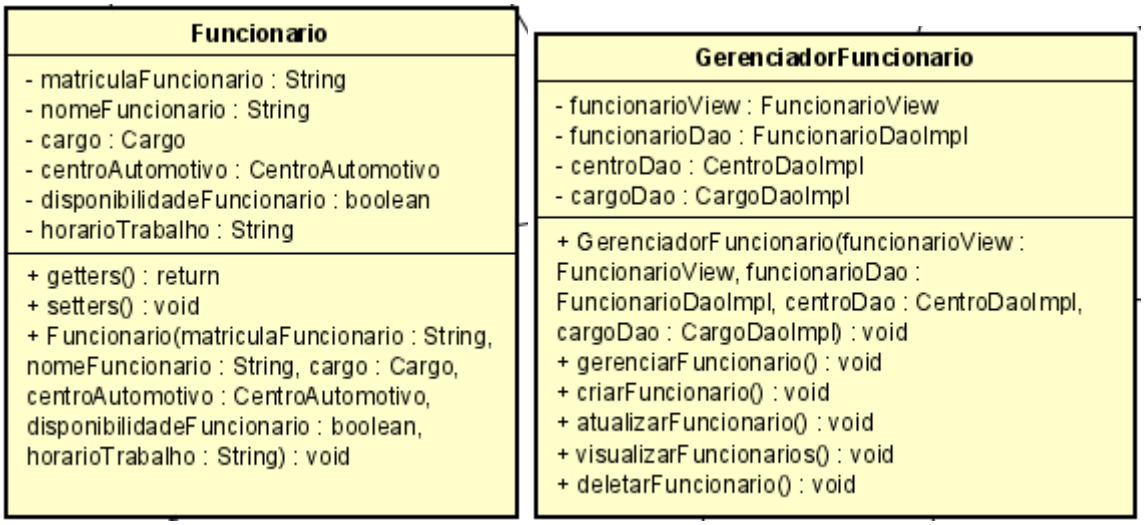
A nomenclatura **MVC (Model-View-Controller)** é um padrão de arquitetura de software amplamente utilizado para desenvolver interfaces de usuário organizadas e escaláveis. Ele separa uma aplicação em três componentes principais: **Model**, **View** e **Controller**, cada uma com responsabilidades distintas. Essa separação facilita a manutenção, o teste e o desenvolvimento de aplicações de maneira mais organizada, modularizada, limpa e deixando cada camada e classe com sua responsabilidade. Aqui está uma visão geral de cada um:

1. **Model (Modelo)**:
    - Representa os dados da aplicação e a lógica de negócios.
    - É responsável por buscar, armazenar e manipular os dados, assim como aplicar regras e operações necessárias.
    - Não interage diretamente com a interface do usuário. Em vez disso, ele notifica a View sobre mudanças nos dados.
2. **View**:
    - É a camada responsável por apresentar os dados ao usuário.
    - Recebe informações do Model e exibe-as de forma amigável, sem saber como os dados são gerados ou armazenados.
    - Captura as entradas do usuário (como cliques ou digitação) e as encaminha ao Controller.
3. **Controller (Controlador)**:
    - Atua como intermediário entre a View e o Model.
    - Recebe as entradas do usuário a partir da View, processa essas entradas (aplicando lógica de negócios), e interage com o Model para buscar ou atualizar os dados.
    - Atualiza a View com base nas respostas do Model.

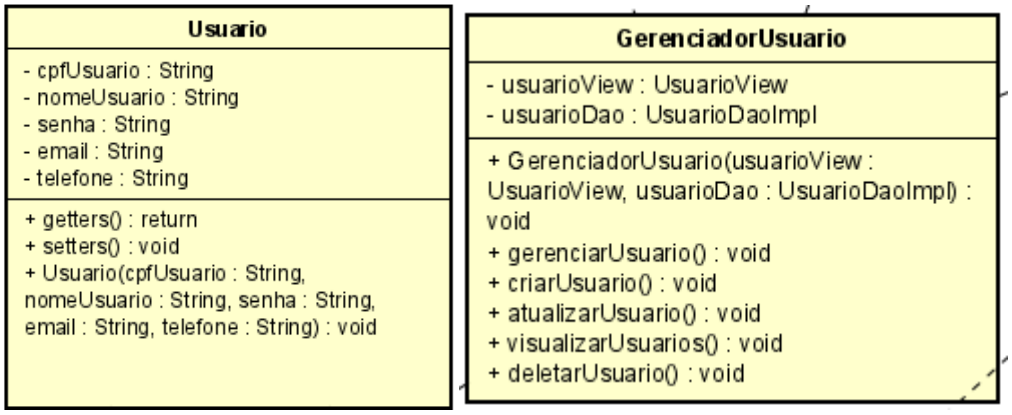
VOs e Gerenciadores/Controllers



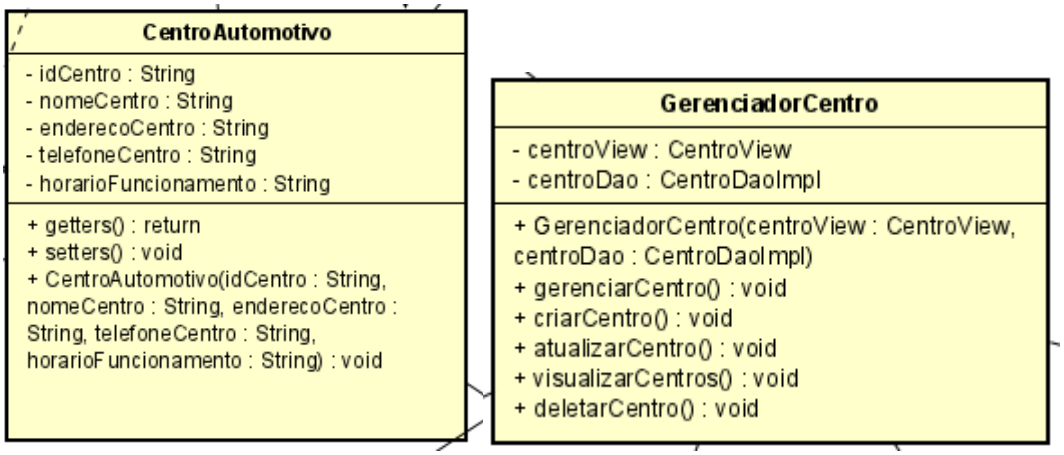
- **Orcamento:** classe Model VO que armazena informações sobre um orçamento (id, descrição, valor total e status). Possui métodos getters e setters e um construtor parametrizado.
- **GerenciadorOrcamento:** gerenciador de Orcamento, responsável por receber as informações obtidas pela View, validar e passar para o DAO, e vice-versa;



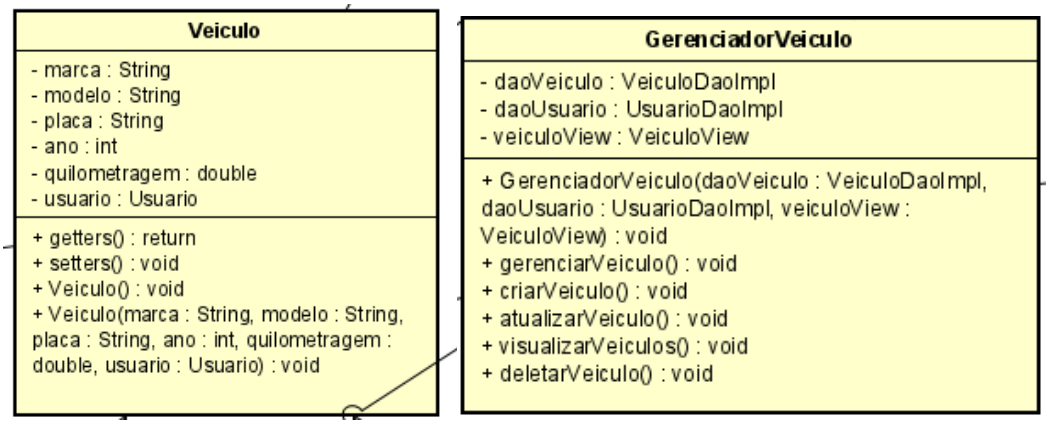
- **Funcionario:** classe Model VO que armazena informações sobre um funcionário (matrícula, nome, cargo, centro automotivo, disponibilidade e horário de trabalho). Possui métodos getters e setters, e um construtor parametrizado;
- **GerenciadorFuncionario:** responsável por receber as informações obtidas pela View, validar e passar para o DAO, e vice-versa;



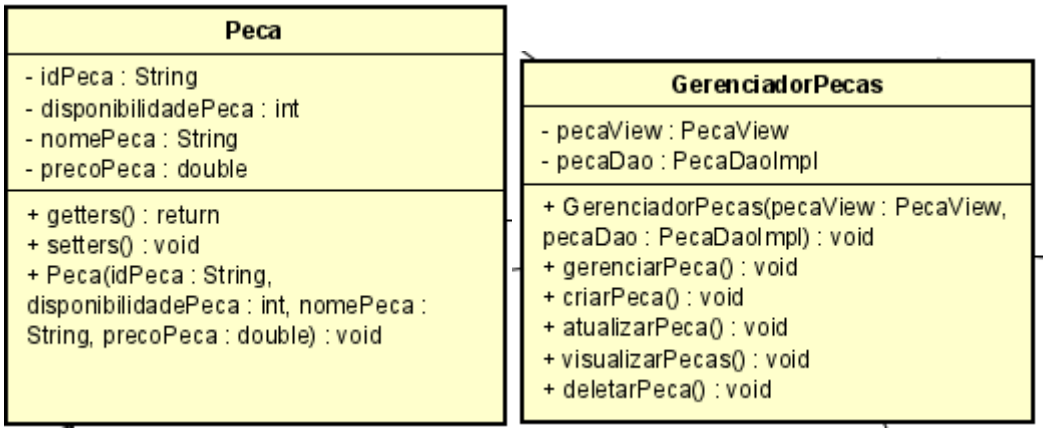
- **Usuario:** classe model VO que armazena informações sobre um usuário (CPF, nome, senha, e-mail, telefone). Possui métodos getters e setters, um construtor parametrizado e um método para imprimir os dados do usuário;
- **GerenciadorUsuario:** responsável por receber as informações obtidas pela View, validar e passar para o DAO, e vice-versa;



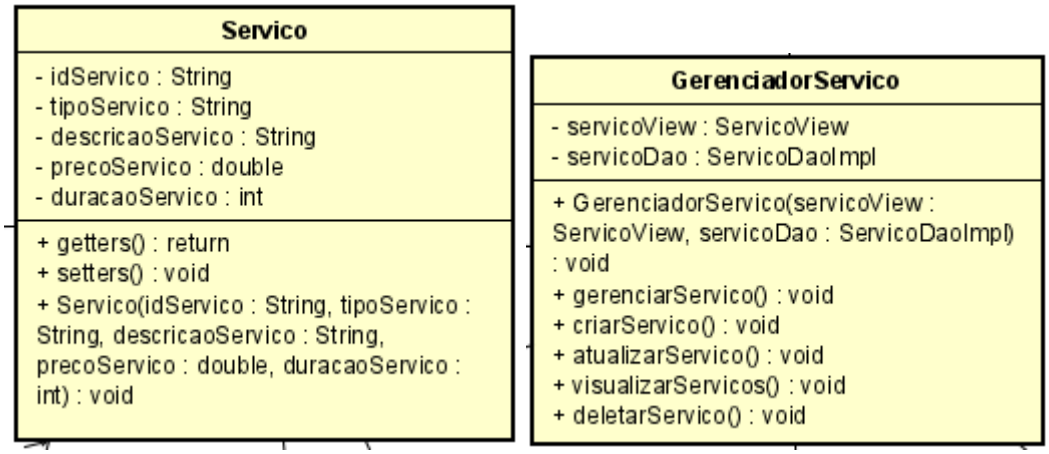
- **CentroAutomotivo:** classe Model VO que armazena informações sobre o centro automotivo (id, nome, endereço, telefone e horário de funcionamento). Possui métodos getters e setters, e construtor parametrizado;
- **GerenciadorCentro:** responsável por receber as informações obtidas pela View, validar e passar para o DAO, e vice-versa;



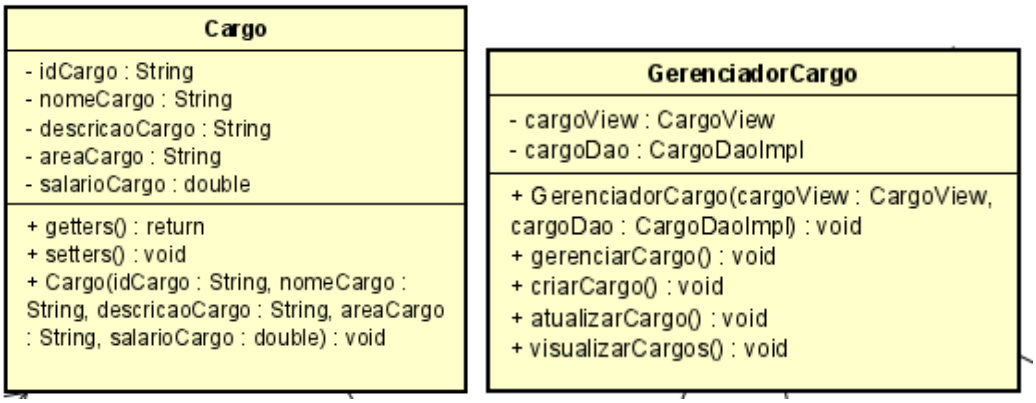
- **Veiculo:** classe Model VO que armazena informações sobre um veículo (marca, modelo, placa, ano e quilometragem). Possui métodos getters e setters, e um construtor parametrizado.
- **GerenciadorVeiculo:** responsável por receber as informações obtidas pela View, validar e passar para o DAO, e vice-versa;



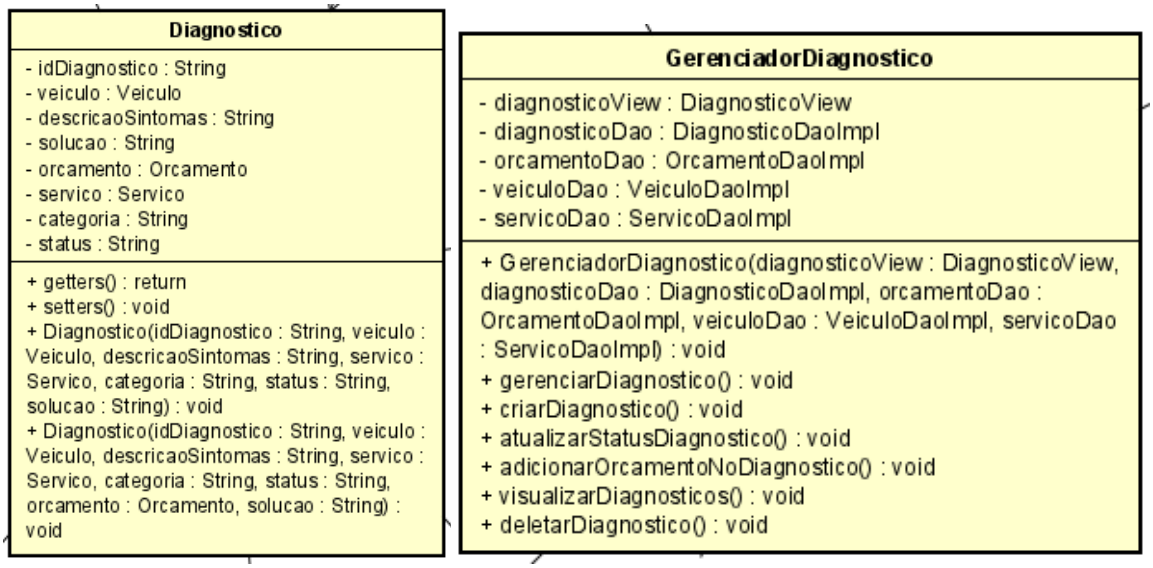
- **Peca:** classe Model VO que armazena informações de uma peça (id, disponibilidade, nome e preço). Possui métodos getters e setters, e um construtor parametrizado;
- **GerenciadorPecas:** responsável por receber as informações obtidas pela View, validar e passar para o DAO, e vice-versa;



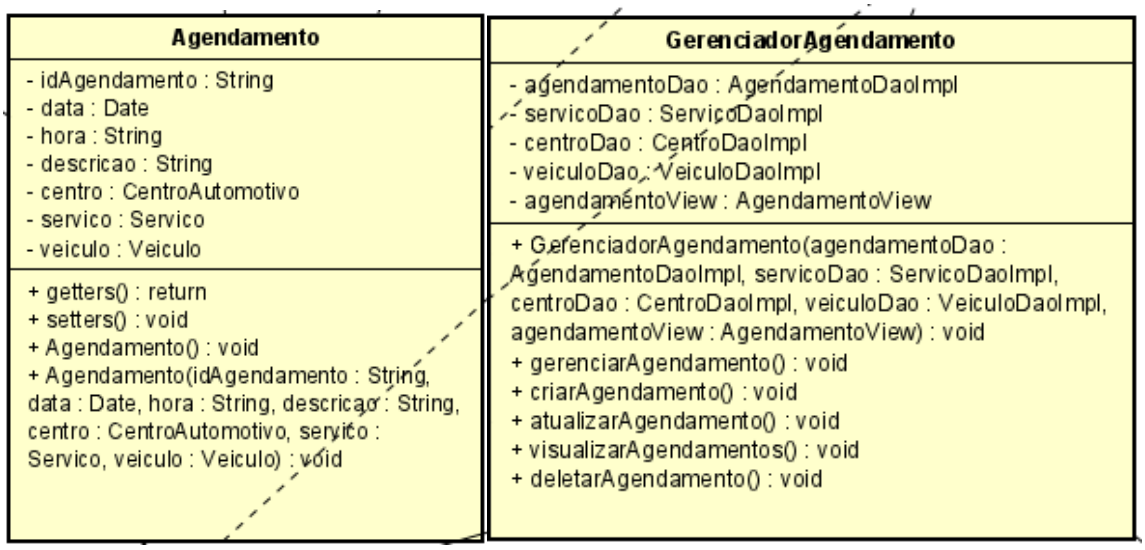
- **Servico:** classe Model VO que armazena informações de um serviço (id, tipo, descrição, preço, duração). Possui métodos getters e setters, e um construtor parametrizado;
- **GerenciadorServico:** responsável por receber as informações obtidas pela View, validar e passar para o DAO, e vice-versa;



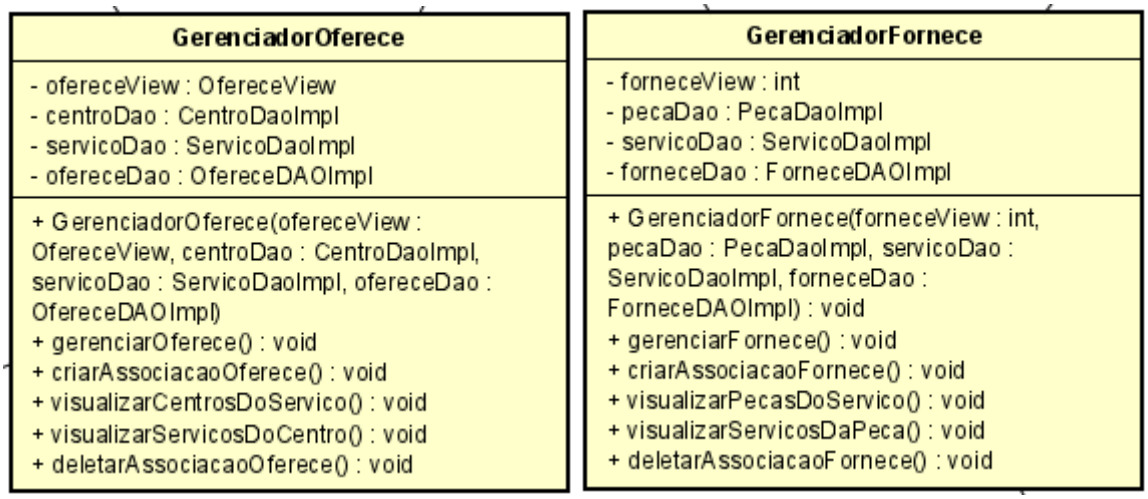
- **Cargo:** classe Model VO que armazena informações de um cargo (id, nome, descrição, área e salário). Possui métodos getters e setters, e um construtor parametrizado;
- **GerenciadorCargo:** responsável por receber as informações obtidas pela View, validar e passar para o DAO, e vice-versa;



- **Diagnostico:** classe Model VO que armazena informações de um diagnóstico (id, veículo, descrição dos sintomas, solução, orçamento, categoria, serviço e status). Possui métodos getters e setters, e dois construtores parametrizados;
- **GerenciadorDiagnostico:** responsável por receber as informações obtidas pela View, validar e passar para o DAO, e vice-versa;



- **Agendamento:** classe Model VO que armazena informações de um agendamento (id, data, hora, descrição, centro automotivo, serviço e veículo). Possui métodos getters e setters, construtor vazio e construtor parametrizado;
- **GerenciadorAgendamento:** responsável por receber as informações obtidas pela View, validar e passar para o DAO, e vice-versa;



- **GerenciadorOferece:** gerenciador das relações entre centro e serviço (tabela associativa). Responsável por receber as informações obtidas pela View, validar e passar para o DAO, e vice-versa;
- **GerenciadorFornece:** gerenciador das relações entre peça e serviço (tabela associativa). Responsável por receber as informações obtidas pela View, validar e passar para o DAO, e vice-versa;

Resumo sobre a camada View

As classes do pacote view são responsáveis pela **interação com o usuário**, seguindo o padrão **MVC**. Sendo assim, uso do **DAO** garante a separação das operações, permitindo que as classes de View se concentrem apenas na interação com o usuário, mantendo assim uma arquitetura mais limpa, mais modular e de mais fácil manutenção.

Camada View

1. AgendamentoView

- **Responsabilidade:** Gerenciar a interação com o usuário no contexto de
- agendamentos.
- **Principais Funcionalidades:**
    - Exibir menus para cadastrar, listar, atualizar ou deletar agendamentos.
    - Imprimir detalhes de um agendamento, como serviço, oficina, data, horário, veículo e usuário.
    - Solicitar informações para cadastro e atualização, como ID, serviço ou veículo.
    - Implementar a interface IView para padronizar a exibição de mensagens e captura de opções do usuário.

2. CargoView

- **Responsabilidade:** Gerenciar a visualização e interação com os cargos do sistema.
- **Principais Funcionalidades:**
    - Exibir menus para cadastro, listagem, atualização e remoção de cargos.
    - Imprimir detalhes de um cargo, como nome e descrição.
    - Solicitar informações necessárias para operações, como atualização e remoção de cargos.
    - Implementar métodos de exibição de mensagens e captura de opções do usuário, seguindo o padrão IView.

3. CentroView

- **Responsabilidade:** Gerenciar a visualização e interação com dados dos centros automotivos.

- **Principais Funcionalidades:**
    - Exibir menus para cadastrar, listar, atualizar e remover centros automotivos.
    - Imprimir informações de centros, como nome, endereço, telefone e horário de funcionamento.
    - Solicitar informações como ID, nome e endereço para as operações.
    - Seguir a padronização da interface IView para interação com o usuário.

4. DiagnosticoView

- **Responsabilidade:** Gerenciar a interação com diagnósticos automotivos no sistema.
- **Principais Funcionalidades:**
    - Exibir menus para cadastrar, listar, atualizar ou deletar diagnósticos.
    - Imprimir detalhes de um diagnóstico, como sintomas, solução, serviço, categoria, status e orçamento.
    - Solicitar entradas como ID do diagnóstico, sintomas, status e ID do serviço.
    - Implementar métodos como exibirMensagem e obterOpcao seguindo a interface IView.
- **Detalhes Técnicos:**
    - Utiliza a classe Diagnostico do pacote model.vo.

5. ForneceView

- **Responsabilidade:** Gerenciar a relação entre peças e serviços no sistema.
- **Principais Funcionalidades:**
    - Exibir menus para cadastrar, listar ou remover relações entre peças e serviços.
    - Imprimir informações detalhadas de peças e serviços, como nome, preço e duração.
    - Solicitar informações, como IDs de peças e serviços, para cadastro e remoção de relações.
    - Implementar a interface IView para garantir a padronização.
- **Detalhes:**
    - Utiliza as classes Peca e Servico do pacote model.vo.

6. FuncionarioView

- **Responsabilidade:** Gerenciar a interação com funcionários do sistema.
- **Principais Funcionalidades:**
    - Exibir menus para cadastrar, listar, atualizar ou deletar funcionários.
    - Apresentar opções de atualização de atributos como nome, horário de trabalho e disponibilidade.
    - Imprimir informações detalhadas de um funcionário, como matrícula, cargo e disponibilidade.
    - Solicitar entradas como matrícula, nome, ID do centro automotivo e ID do cargo.
    - Implementar métodos de exibição e captura de dados padronizados pela interface IView.
- **Detalhes:**
    - Utiliza a classe Funcionario do pacote model.vo.

7. IView

- **Responsabilidade:** Definir um contrato para as views do sistema, padronizando a interação com o usuário.
- **Principais Funcionalidades:**
    - exibirMensagem: Método para exibir mensagens ao usuário.
    - obterOpcao: Método para capturar a escolha do usuário no menu.

8. MenuPrincipalView

- **Responsabilidade:** Gerenciar a navegação principal do sistema, permitindo o acesso a funcionalidades como gerenciamento de usuários, veículos, peças, serviços, cargos e centros automotivos.
- **Principais Funcionalidades:**
    - Apresentar um menu principal com as opções de gerenciamento.
    - Coletar e processar a escolha do usuário.

- Redirecionar para a view correspondente.
- Oferecer a opção de sair do sistema.

9. OfereceView

- **Responsabilidade:** Gerenciar a relação entre centros automotivos e serviços oferecidos.
- **Principais Funcionalidades:**
  - Criar associações entre centros automotivos e serviços.
  - Listar as associações existentes.
  - Deletar relações entre centros e serviços.
  - Coletar IDs dos centros e serviços para realizar as operações.

10. OrcamentoView

- **Responsabilidade:** Gerenciar orçamentos automotivos, com base em diagnósticos e serviços prestados.
- **Principais Funcionalidades:**
  - Criar novos orçamentos associando diagnósticos, descrição e valor total.
  - Listar orçamentos existentes no sistema.
  - Atualizar o status dos orçamentos.
  - Deletar orçamentos, removendo-os do sistema.

11. PecaView

- **Responsabilidade:** Gerenciar o cadastro e a manutenção de peças automotivas.
- **Principais Funcionalidades:**
  - Cadastrar novas peças com nome, preço e disponibilidade.
  - Listar todas as peças cadastradas.
  - Atualizar informações de peças, como nome, preço e disponibilidade.
  - Deletar peças do sistema.

12. ServicoView

- **Responsabilidade:** Gerenciar os serviços automotivos oferecidos pelos centros.
- **Principais Funcionalidades:**
    - Cadastrar novos serviços, fornecendo tipo, descrição, preço e duração.
    - Listar serviços existentes no sistema.
    - Atualizar informações de serviços.
    - Deletar serviços do sistema.

13. UsuarioView

- **Responsabilidade:** Gerenciar o cadastro, listagem e manutenção dos usuários do sistema.
- **Principais Funcionalidades:**
    - Cadastrar novos usuários solicitando CPF, nome, e-mail, telefone e senha.
    - Listar todos os usuários registrados.
    - Atualizar informações de um usuário existente.
    - Deletar usuários do sistema.

14. VeiculoView

- **Responsabilidade:** Gerenciar a interação com o usuário relacionada aos veículos no sistema.
- **Principais Funcionalidades:**
    - **Exibir Menu de Veículo:** Apresenta opções para cadastrar, listar, atualizar ou deletar veículos.
    - **Exibir Menu de Atualização de Veículo:** Oferece opções para atualizar atributos do veículo, como marca, modelo, ano, quilometragem e CPF do proprietário.
    - **Imprimir Informações do Veículo:** Exibe detalhadamente os dados de um veículo específico, como placa, marca, modelo, ano, quilometragem e CPF do proprietário.
    - **Solicitar Entradas do Usuário:** Coleta informações como marca, modelo, placa, ano, quilometragem e CPF do proprietário para as operações de cadastro e atualização.

- **Métodos de Interação:** Implementa métodos como exibirMensagem para mostrar mensagens ao usuário e obterOpcao para capturar a escolha do usuário no menu.

Resumo do Model DAO

O padrão **DAO (Data Access Object)** é uma das principais abordagens utilizadas em projetos de software para separar a lógica de acesso a dados da lógica de negócios. Ele fornece uma abstração que permite que os objetos da aplicação interajam com a camada de banco de dados de forma isolada, encapsulada sem se preocupar com os detalhes de como os dados são armazenados ou recuperados, facilitando a manutenção de dados.

Resumo da função de cada classe presente dentro de dao que são responsáveis pela interação com o banco de dados para realizar operações CRUD (Create, Read, Update, Delete)

Package **agendamento:**

- Gerencia o acesso a dados relacionados aos agendamentos no sistema.

1. AgendamentoDAO (Interface)

Uma interface que define a estrutura para a manipulação dos dados de agendamentos no banco de dados. As operações básicas incluem:

- **inserir**: Adiciona um novo agendamento.
- **listar**: Recupera todos os agendamentos.
- **buscarPorId**: Encontra um agendamento específico pelo ID.
- **agendamentoExiste**: Verifica se um agendamento existe com base no ID.
- **atualizarDataAgendamento**: Atualiza a data de um agendamento existente.
- **atualizarHorarioAgendamento**: Atualiza o horário de um agendamento.
- **atualizarDescricaoAgendamento**: Atualiza a descrição de um agendamento.
- **atualizarCentro**: Atualiza o centro automotivo associado a um agendamento.
- **atualizarServico**: Atualiza o serviço associado a um agendamento.

- **atualizarVeiculo**: Atualiza o veículo associado a um agendamento.
- **deletar**: Remove um agendamento baseado no ID.

2. AgendamentoDaoImpl (Classe)

Esta classe implementa a interface AgendamentoDAO e fornece a lógica para manipular agendamentos no banco de dados. Aqui estão os detalhes principais:

- **Conexão com o banco de dados**: Utiliza OracleDataSource para conectar-se ao banco de dados Oracle usando as credenciais fornecidas.
- **Métodos**:
    - **inserir**: Insere um novo agendamento na tabela agendamento.
    - **listar**: Recupera todos os agendamentos e constrói objetos Agendamento com dados do banco e informações adicionais, como CentroAutomotivo, Servico, e Veiculo.
    - **buscarPorId**: Recupera um agendamento específico com base no ID e cria um objeto Agendamento.
    - **agendamentoExiste**: Verifica a existência de um agendamento com um dado ID.
    - **atualizarDataAgendamento**: Atualiza a data de um agendamento existente.
    - **atualizarHorarioAgendamento**: Atualiza o horário de um agendamento existente.
    - **atualizarDescricaoAgendamento**: Atualiza a descrição de um agendamento existente.
    - **atualizarCentro**: Atualiza o centro automotivo de um agendamento se o centro existir.
    - **atualizarServico**: Atualiza o serviço de um agendamento se o serviço existir.
    - **atualizarVeiculo**: Atualiza o veículo de um agendamento se o veículo existir.
    - **deletar**: Remove um agendamento se ele existir.

Package **cargo:**

- Responsável por operações de persistência relacionadas aos cargos dos funcionários, como criar, listar, atualizar e remover cargos.

1. CargoDAO (Interface)

Esta interface define os métodos que uma implementação deve fornecer para manipulação dos dados de cargos em um banco de dados. Os métodos são:

- **inserir**: Adiciona um novo cargo ao banco de dados.
- **listar**: Recupera todos os cargos.
- **buscarPorId**: Encontra um cargo específico pelo ID.
- **cargoExiste**: Verifica se um cargo existe com base no ID.
- **atualizarNomeCargo**: Atualiza o nome de um cargo existente.
- **atualizarDescricaoCargo**: Atualiza a descrição de um cargo existente.
- **atualizarAreaCargo**: Atualiza a área de um cargo existente.
- **atualizarSalarioCargo**: Atualiza o salário de um cargo existente.
- **deletar**: Remove um cargo baseado no ID.

2. CargoDaoImpl (Classe)

Esta classe implementa a interface CargoDAO e fornece a implementação concreta para manipular os dados de cargos no banco de dados. Aqui estão os detalhes principais:

- **Conexão com o Banco de Dados**: Utiliza OracleDataSource para conectar-se ao banco de dados Oracle usando credenciais específicas.
- **Métodos**:
    - **inserir**: Insere um novo cargo na tabela cargo com os campos id_cargo, nome_cargo, descricao_cargo, area_cargo, e salario_cargo.
    - **listar**: Recupera todos os cargos da tabela cargo e cria objetos Cargo com os dados recuperados.
    - **buscarPorId**: Recupera um cargo específico com base no ID e cria um objeto Cargo com os dados encontrados.
    - **cargoExiste**: Verifica se um cargo existe com base no ID fornecido, retornando true se existir e false caso contrário.
    - **atualizarNomeCargo**: Atualiza o nome de um cargo se o ID existir no banco de dados.
    - **atualizarDescricaoCargo**: Atualiza a descrição de um cargo se o ID existir no banco de dados.
    - **atualizarAreaCargo**: Atualiza a área de um cargo se o ID existir no banco de dados.

- **atualizarSalarioCargo**: Atualiza o salário de um cargo se o ID existir no banco de dados.
- **deletar**: Remove um cargo se o ID existir no banco de dados.

Package **centroAutomotivo:**

- Contém a lógica de acesso a dados para os centros automotivos, permitindo gerenciar informações como nome, endereço, telefone, etc.

1. Interface CentroDAO

A interface CentroDAO define um contrato para operações de acesso a dados para centros automotivos. É uma interface genérica que estabelece métodos para criar, ler, atualizar e deletar (CRUD) registros de centros automotivos.

Métodos:

- **inserir(CentroAutomotivo centroAutomotivo)**: Insere um novo centro automotivo no banco de dados.
- **listar()**: Retorna uma lista de todos os centros automotivos presentes no banco de dados.
- **buscarPorId(String id)**: Busca e retorna um centro automotivo específico com base no ID fornecido.
- **centroExiste(String idCentro)**: Verifica a existência de um centro automotivo pelo ID fornecido.
- **atualizarNomeCentro(String idCentro, String nome)**: Atualiza o nome de um centro automotivo existente.
- **atualizarEndereco(String idCentro, String endereco)**: Atualiza o endereço de um centro automotivo existente.
- **atualizarTelefoneCentro(String idCentro, String telefone)**: Atualiza o telefone de um centro automotivo existente.
- **atualizarHorarioFuncionamento(String idCentro, String horario)**: Atualiza o horário de funcionamento de um centro automotivo existente.
- **deletar(String id)**: Remove um centro automotivo do banco de dados com base no ID fornecido.

2. Classe CentroDaoImpl

A classe CentroDaoImpl implementa a interface CentroDAO e fornece a implementação concreta dos métodos para acessar e manipular dados de centros automotivos em um banco de dados Oracle. Utiliza JDBC para a comunicação com o banco de dados.

Aspectos principais da implementação:

- **Conexão com o Banco de Dados**:
  - Utiliza OracleDataSource para criar uma conexão com o banco de dados Oracle.
  - As credenciais (usuário e senha) são obtidas da classe Credenciais.
- **Métodos Implementados**:
  - **inserir(CentroAutomotivo ca)**: Executa um comando SQL INSERT para adicionar um novo centro automotivo à tabela centro_automotivo. Verifica se a inserção foi bem-sucedida com base no número de linhas afetadas.
  - **listar()**: Executa um comando SQL SELECT para recuperar todos os registros da tabela centro_automotivo. Converte os resultados em objetos CentroAutomotivo e os adiciona a uma lista.
  - **buscarPorId(String idCentro)**: Executa um comando SQL SELECT para encontrar um centro automotivo específico pelo ID. Cria e retorna um objeto CentroAutomotivo com os dados recuperados.
  - **centroExiste(String idCentro)**: Executa um comando SQL SELECT COUNT(*) para verificar a existência de um centro automotivo com o ID fornecido. Retorna true se o número de registros for maior que zero.
  - **atualizarNomeCentro(String idCentro, String nome)**: Atualiza o nome do centro automotivo especificado. Executa um comando SQL UPDATE se o ID existir.
  - **atualizarEndereco(String idCentro, String endereco)**: Atualiza o endereço do centro automotivo especificado. Executa um comando SQL UPDATE se o ID existir.
  - **atualizarTelefoneCentro(String idCentro, String telefone)**: Atualiza o telefone do centro automotivo especificado. Executa um comando SQL UPDATE se o ID existir.
  - **atualizarHorarioFuncionamento(String idCentro, String horario)**: Atualiza o horário de funcionamento do centro automotivo especificado. Executa um comando SQL UPDATE se o ID existir.

- **deletar(String id)**: Remove um centro automotivo da tabela centro_automotivo com o ID fornecido. Executa um comando SQL DELETE se o ID existir.

Package **credenciais:**

- Lida com as credenciais de login dos usuários no banco de dados.

Package **diagnostico:**

- Gerencia o acesso aos dados relacionados aos diagnósticos automotivos.

1. Interface DiagnosticoDAO

A interface DiagnosticoDAO define os métodos essenciais para realizar operações CRUD (Create, Read, Update, Delete) sobre os diagnósticos em um banco de dados. Serve como um contrato para qualquer classe que implemente a manipulação de dados de diagnósticos.

Métodos definidos:

- **inserir(Diagnostico diagnostico)**: Insere um novo diagnóstico no banco de dados.
- **listar()**: Retorna uma lista de todos os diagnósticos presentes no banco de dados.
- **buscarPorId(String id)**: Busca e retorna um diagnóstico específico com base no ID fornecido.
- **diagnosticoExiste(String idDiagnostico)**: Verifica a existência de um diagnóstico pelo ID fornecido.
- **atualizarStatusDiagnostico(String idDiagnostico, String status)**: Atualiza o status de um diagnóstico existente.
- **deletar(String id)**: Remove um diagnóstico do banco de dados com base no ID fornecido.

2. Classe DiagnosticoDaoImpl

A classe DiagnosticoDaoImpl fornece a implementação concreta da interface DiagnosticoDAO, utilizando JDBC para realizar operações sobre o banco de dados Oracle. Esta classe gerencia a conexão com o banco de dados e executa comandos SQL para inserir, listar, buscar, atualizar e deletar diagnósticos.

Aspectos principais da implementação:

- **Conexão com o Banco de Dados**:
  - A classe utiliza OracleDataSource para criar uma conexão com o banco de dados Oracle.

  - As credenciais de conexão (usuário e senha) são obtidas da classe Credenciais.

- **Dependências**:
  - **VeiculoDaoImpl**: Usada para recuperar informações sobre veículos associados a diagnósticos.
  - **ServicoDaoImpl**: Usada para recuperar informações sobre serviços associados a diagnósticos.
  - **OrcamentoDaoImpl**: Usada para recuperar informações sobre orçamentos associados a diagnósticos.
- **Métodos**:
  - **inserir(Diagnostico d)**: Insere um novo diagnóstico na tabela diagnostico do banco de dados. O método prepara um comando SQL INSERT e executa-o, verificando se a operação foi bem-sucedida com base no número de linhas afetadas.
  - **listar()**: Recupera todos os diagnósticos da tabela diagnostico. Para cada registro, busca detalhes adicionais sobre o veículo, serviço e orçamento associados. Cria objetos Diagnostico e os adiciona a uma lista retornada ao final.
  - **buscarPorId(String id)**: Recupera um diagnóstico específico pelo ID. O método prepara e executa um comando SQL SELECT, cria um objeto Diagnostico com as informações recuperadas e retorna esse objeto.
  - **diagnosticoExiste(String idDiagnostico)**: Verifica se um diagnóstico com o ID fornecido existe na tabela diagnostico. Executa um comando SQL SELECT COUNT(*) e retorna true se o número de registros for maior que zero.
  - **atualizarStatusDiagnostico(String idDiagnostico, String status)**: Atualiza o status de um diagnóstico existente. O método prepara e executa um comando SQL UPDATE se o diagnóstico existir.
  - **inserirOrcamentoNoDiagnostico(String idDiagnostico, String idOrcamento)**: Atualiza o diagnóstico especificado para associar um orçamento. Executa um comando SQL UPDATE para definir o campo orcamento_id_orcamento no registro do diagnóstico.
  - **deletar(String id)**: Remove um diagnóstico da tabela diagnostico com o ID fornecido. O método executa um comando SQL DELETE se o diagnóstico existir.

Package **fornece:**

- Gerencia a relação entre peças e serviços, registrando quais peças são fornecidas para quais serviços automotivos.

1 1. Interface ForneceDAO

A interface ForneceDAO define os métodos necessários para manipular as associações entre peças e serviços no banco de dados. É uma interface que facilita o gerenciamento de relacionamentos entre peças e serviços em um sistema de gerenciamento.

Métodos definidos:

- **adicionarAssociacao(String id_peca, String id_servico)**: Adiciona uma nova associação entre uma peça e um serviço. Retorna true se a associação for criada com sucesso e false caso contrário.
- **deletarAssociacao(String id_peca, String id_servico)**: Remove uma associação existente entre uma peça e um serviço. Retorna true se a associação for deletada com sucesso e false caso contrário.
- **listarServicosPorPeca(String id_peca)**: Lista todos os serviços associados a uma peça específica. Retorna uma lista de IDs de serviços.
- **listarPecasPorServico(String id_servico)**: Lista todas as peças associadas a um serviço específico. Retorna uma lista de IDs de peças.

2 2. Classe ForneceDAOImpl

A classe ForneceDAOImpl fornece a implementação concreta da interface ForneceDAO, utilizando JDBC para realizar operações sobre o banco de dados Oracle. Esta classe gerencia a conexão com o banco de dados e executa comandos SQL para manipular as associações entre peças e serviços.

Aspectos principais da implementação:

- **Conexão com o Banco de Dados**:
  - A classe usa OracleDataSource para estabelecer uma conexão com o banco de dados Oracle.
  - As credenciais de conexão (usuário e senha) são obtidas da classe Credenciais.

- **Métodos**:
    - **adicionarAssociacao(String id_peca, String id_servico)**: Adiciona uma nova associação entre uma peça e um serviço na tabela fornece. O método prepara um comando SQL INSERT e executa-o. Retorna true se a operação for bem-sucedida e false caso contrário. Caso ocorra um erro (por exemplo, a peça e o serviço já estejam associados), é exibida uma mensagem de erro.
    - **deletarAssociacao(String id_peca, String id_servico)**: Remove uma associação existente entre uma peça e um serviço na tabela fornece. Executa um comando SQL DELETE para remover a associação. Retorna true se a operação for bem-sucedida e false em caso de erro. Em caso de erro, a mensagem de erro é exibida.
    - **listarServicosPorPeca(String id_peca)**: Recupera todos os IDs de serviços associados a uma peça específica. Executa um comando SQL SELECT e adiciona os IDs de serviços retornados a uma lista. Retorna a lista de serviços associados à peça. Em caso de erro, a mensagem de erro é exibida.
    - **listarPecasPorServico(String id_servico)**: Recupera todos os IDs de peças associadas a um serviço específico. Executa um comando SQL SELECT e adiciona os IDs de peças retornados a uma lista. Retorna a lista de peças associadas ao serviço. Em caso de erro, a mensagem de erro é exibida.

Package **funcionario:**

- Contém as operações de persistência de dados dos funcionários do sistema, permitindo criar, listar, atualizar e remover registros de funcionários.

1. Interface FuncionarioDAO

A interface FuncionarioDAO define os métodos necessários para realizar operações CRUD (Create, Read, Update, Delete) relacionadas aos funcionários no banco de dados.

Métodos definidos:

- **inserir(Funcionario funcionario)**: Insere um novo registro de funcionário no banco de dados. Retorna true se a inserção for bem-sucedida e false caso contrário.
- **listar()**: Lista todos os funcionários presentes no banco de dados. Retorna uma lista de objetos Funcionario.
- **buscarPorId(String id)**: Busca um funcionário pelo seu ID (matrícula). Retorna um objeto Funcionario se encontrado, ou null caso contrário.

- **funcionarioExiste(String idFuncionario)**: Verifica se um funcionário com a matrícula fornecida existe no banco de dados. Retorna true se existir e false caso contrário.
- **atualizarNomeFuncionario(String matricula, String nome)**: Atualiza o nome de um funcionário específico. Caso a matrícula não exista, uma mensagem é exibida indicando que o funcionário não está registrado.
- **atualizarHorarioTrabalho(String matricula, String horario)**: Atualiza o horário de trabalho de um funcionário específico. Caso a matrícula não exista, uma mensagem é exibida indicando que o funcionário não está registrado.
- **atualizarDisponibilidadeFuncionario(String matricula, boolean disponibilidade)**: Atualiza a disponibilidade de um funcionário específico. Caso a matrícula não exista, uma mensagem é exibida indicando que o funcionário não está registrado.
- **atualizarCentro(String matricula, String idCentro)**: Atualiza o centro automotivo associado a um funcionário. Caso a matrícula do funcionário ou o ID do centro não existam, uma mensagem é exibida indicando que o funcionário ou o centro não estão registrados.
- **atualizarCargo(String matricula, String idCargo)**: Atualiza o cargo de um funcionário específico. Caso a matrícula do funcionário ou o ID do cargo não existam, uma mensagem é exibida indicando que o funcionário ou o cargo não estão registrados.
- **deletar(String matricula)**: Remove um funcionário do banco de dados com base na matrícula fornecida. Caso a matrícula não exista, uma mensagem é exibida indicando que o funcionário não está registrado.

2. Classe FuncionarioDaoImpl

A classe FuncionarioDaoImpl é a implementação concreta da interface FuncionarioDAO, realizando operações de banco de dados usando JDBC para manipular registros de funcionários. Ela também interage com as classes CentroDaoImpl e CargoDaoImpl para buscar informações relacionadas ao centro automotivo e ao cargo.

- **Conexão com o Banco de Dados**:
    - Utiliza OracleDataSource para conectar-se ao banco de dados Oracle, configurado com as credenciais obtidas da classe Credenciais.
- **Métodos**:
    - **inserir(Funcionario f)**: Insere um novo funcionário na tabela funcionario. O método prepara e executa um comando SQL INSERT, retornando true se a operação for bem-sucedida e false em caso de erro.

- **listar()**: Recupera todos os funcionários da tabela funcionario. Executa um comando SQL SELECT e cria uma lista de objetos Funcionario com base nos dados recuperados. Utiliza CargoDaoImpl e CentroDaoImpl para buscar informações adicionais sobre cargo e centro.
- **buscarPorId(String id)**: Recupera um funcionário específico da tabela funcionario com base na matrícula. Se encontrado, retorna um objeto Funcionario criado a partir dos dados recuperados; caso contrário, retorna null.
- **funcionarioExiste(String matricula)**: Verifica a existência de um funcionário na tabela funcionario com a matrícula fornecida. Executa um comando SQL SELECT COUNT(*) e retorna true se o contador for maior que zero.
- **atualizarNomeFuncionario(String matricula, String nome)**: Atualiza o nome de um funcionário com base na matrícula fornecida. O método só executa a atualização se o funcionário existir. Exibe mensagens de sucesso ou erro conforme o resultado.
- **atualizarHorarioTrabalho(String matricula, String horario)**: Atualiza o horário de trabalho de um funcionário. Só realiza a atualização se o funcionário existir. Mensagens de sucesso ou erro são exibidas conforme o resultado.
- **atualizarDisponibilidadeFuncionario(String matricula, boolean disponibilidade)**: Atualiza a disponibilidade de um funcionário. Verifica a existência do funcionário antes de realizar a atualização. Mensagens apropriadas são exibidas para informar sobre o sucesso ou falha da operação.
- **atualizarCentro(String matricula, String idCentro)**: Atualiza o centro automotivo associado a um funcionário. Verifica se o funcionário e o centro existem antes de realizar a atualização. Mensagens são exibidas para indicar o sucesso ou falha da operação.
- **atualizarCargo(String matricula, String idCargo)**: Atualiza o cargo de um funcionário. Verifica a existência do funcionário e do cargo antes de realizar a atualização. Mensagens apropriadas são exibidas para informar sobre o resultado da operação.
- **deletar(String matricula)**: Remove um funcionário do banco de dados com base na matrícula. Só realiza a remoção se o funcionário existir. Mensagens são exibidas para indicar o sucesso ou falha da operação.

Package **oferece:**

- Lida com a relação entre os centros automotivos e os serviços.

1. Interface OfereceDAO

A interface OfereceDAO define os métodos necessários para gerenciar as associações entre serviços e centros automotivos no banco de dados.

Métodos:

- **adicionarAssociacao(String id_servico, String id_centro)**: Adiciona uma associação entre um serviço e um centro automotivo. Retorna true se a operação for bem-sucedida e false em caso de falha (por exemplo, se a associação já existir).
- **deletarAssociacao(String id_servico, String id_centro)**: Remove uma associação existente entre um serviço e um centro automotivo. Retorna true se a operação for bem-sucedida e false em caso de falha.
- **listarServicosPorCentro(String id_centro)**: Lista todos os serviços associados a um centro automotivo específico. Retorna uma lista de IDs de serviços.
- **listarCentrosPorServico(String id_servico)**: Lista todos os centros automotivos associados a um serviço específico. Retorna uma lista de IDs de centros automotivos.

2. Classe OfereceDAOImpl

A classe OfereceDAOImpl é a implementação concreta da interface OfereceDAO, que realiza operações de banco de dados para gerenciar as associações entre serviços e centros automotivos usando JDBC.

- **Conexão com o Banco de Dados**:
  - Utiliza OracleDataSource para conectar-se ao banco de dados Oracle, configurado com as credenciais obtidas da classe Credenciais.
- **Métodos**:
  - **adicionarAssociacao(String id_servico, String id_centro)**: Adiciona uma nova associação entre um serviço e um centro automotivo na tabela oferece. Utiliza um comando SQL INSERT. Retorna true se a inserção for bem-sucedida e false se a associação já existir ou se ocorrer um erro.
  - **deletarAssociacao(String id_servico, String id_centro)**: Remove uma associação existente entre um serviço e um centro automotivo da tabela oferece. Utiliza um comando SQL DELETE. Retorna true se a remoção for bem-sucedida e false se ocorrer um erro.
  - **listarServicosPorCentro(String id_centro)**: Recupera todos os serviços associados a um centro automotivo específico. Executa um comando SQL

SELECT e adiciona os IDs de serviços encontrados a uma lista. Retorna essa lista.

- **listarCentrosPorServico(String id_servico)**: Recupera todos os centros automotivos associados a um serviço específico. Executa um comando SQL SELECT e adiciona os IDs de centros encontrados a uma lista. Retorna essa lista.

Package **orcamento:**

É responsável pela gestão de orçamentos no sistema, incluindo a criação, atualização, busca e exclusão de orçamentos. Contém a interface OrcamentoDAO e a classe concreta OrcamentoDaoImpl.

1. Interface OrcamentoDAO

A interface OrcamentoDAO define os métodos que devem ser implementados para realizar operações sobre orçamentos no banco de dados.

Métodos:

- **inserir(Orcamento orcamento) throws SQLException**: Insere um novo orçamento no banco de dados. Recebe um objeto Orcamento e lança uma SQLException em caso de erro.
- **listar() throws SQLException**: Lista todos os orçamentos presentes no banco de dados. Retorna uma lista de objetos Orcamento e lança uma SQLException em caso de erro.
- **buscarPorId(String id) throws SQLException**: Busca um orçamento pelo seu ID. Recebe um ID e retorna um objeto Orcamento. Lança uma SQLException em caso de erro.
- **orcamentoExiste(String idOrcamento) throws SQLException**: Verifica se um orçamento com o ID especificado existe no banco de dados. Recebe um ID e retorna true se o orçamento existir e false caso contrário. Lança uma SQLException em caso de erro.
- **atualizarStatus(String idOrcamento, String status) throws SQLException**: Atualiza o status de um orçamento existente. Recebe o ID do orçamento e o novo status, e lança uma SQLException em caso de erro.
- **deletar(String id) throws SQLException**: Remove um orçamento pelo seu ID. Recebe um ID e lança uma SQLException em caso de erro.

2. Classe OrcamentoDaoImpl

A classe OrcamentoDaoImpl é a implementação concreta da interface OrcamentoDAO, realizando operações no banco de dados Oracle usando JDBC.

- **Conexão com o Banco de Dados**:
    - Utiliza OracleDataSource para estabelecer uma conexão com o banco de dados Oracle. As credenciais de conexão são fornecidas pela classe Credenciais.
- **Métodos**:
    - **inserir(Orcamento orcamento)**: Adiciona um novo orçamento à tabela orcamento. Utiliza um comando SQL INSERT e retorna true se a operação for bem-sucedida e false se ocorrer um erro.
    - **listar()**: Recupera todos os orçamentos da tabela orcamento. Executa um comando SQL SELECT, cria objetos Orcamento com os dados retornados e os adiciona a uma lista. Retorna essa lista.
    - **buscarPorId(String id)**: Recupera um orçamento específico com base no seu ID. Executa um comando SQL SELECT e cria um objeto Orcamento com os dados retornados. Retorna o objeto ou null se não encontrar o orçamento.
    - **orcamentoExiste(String idOrcamento)**: Verifica se um orçamento com o ID especificado existe na tabela orcamento. Executa um comando SQL SELECT COUNT(*) e retorna true se o orçamento existir e false caso contrário.
    - **atualizarStatus(String idOrcamento, String status)**: Atualiza o status de um orçamento existente. Executa um comando SQL UPDATE e imprime uma mensagem de sucesso se a atualização for bem-sucedida ou uma mensagem de erro caso contrário.
    - **deletar(String id)**: Remove um orçamento com o ID especificado da tabela orcamento. Executa um comando SQL DELETE e imprime uma mensagem de sucesso se a remoção for bem-sucedida ou uma mensagem de erro caso contrário.

Package **peca:**

Este pacote é responsável pela gestão de peças no sistema, incluindo a inserção, atualização, busca e exclusão de peças. Contém a interface PecaDAO e a classe concreta PecaDaoImpl.

1. Interface PecaDAO

A interface PecaDAO define os métodos que devem ser implementados para realizar operações sobre peças no banco de dados.

Métodos definidos:

- **inserir(Peca peca)**: Insere uma nova peça no banco de dados. Recebe um objeto Peca e retorna true se a inserção for bem-sucedida e false caso contrário.
- **listar()**: Lista todas as peças presentes no banco de dados. Retorna uma lista de objetos Peca.
- **buscarPorId(String id)**: Busca uma peça pelo seu ID. Recebe um ID e retorna um objeto Peca. Se a peça não for encontrada, retorna null.
- **pecaExiste(String idPeca)**: Verifica se uma peça com o ID especificado existe no banco de dados. Recebe um ID e retorna true se a peça existir e false caso contrário.
- **atualizarDisponibilidade(String idPeca, int disponibilidade)**: Atualiza a disponibilidade de uma peça existente. Recebe o ID da peça e a nova quantidade de disponibilidade.
- **atualizarNomePeca(String idPeca, String nome)**: Atualiza o nome de uma peça existente. Recebe o ID da peça e o novo nome.
- **atualizarPrecoPeca(String idPeca, double preco)**: Atualiza o preço de uma peça existente. Recebe o ID da peça e o novo preço.
- **deletar(String id)**: Remove uma peça pelo seu ID. Recebe um ID e realiza a exclusão da peça correspondente no banco de dados.

2. Classe PecaDaoImpl

A classe PecaDaoImpl é a implementação concreta da interface PecaDAO, realizando operações no banco de dados Oracle usando JDBC.

Conexão com o Banco de Dados:

  - Utiliza OracleDataSource para estabelecer uma conexão com o banco de dados Oracle. As credenciais de conexão são fornecidas pela classe Credenciais.

- **Métodos**:

  - **inserir(Peca peca)**: Adiciona uma nova peça à tabela peca. Utiliza um comando SQL INSERT e retorna true se a operação for bem-sucedida, e false em caso de erro.

- **listar()**: Recupera todas as peças da tabela peca. Executa um comando SQL SELECT, cria objetos Peca com os dados retornados e os adiciona a uma lista. Retorna essa lista.
- **buscarPorId(String id)**: Recupera uma peça específica com base no seu ID. Executa um comando SQL SELECT e cria um objeto Peca com os dados retornados. Retorna o objeto ou null se não encontrar a peça.
- **pecaExiste(String idPeca)**: Verifica se uma peça com o ID especificado existe na tabela peca. Executa um comando SQL SELECT COUNT(*) e retorna true se a peça existir e false caso contrário.
- **atualizarDisponibilidade(String idPeca, int disponibilidade)**: Atualiza a quantidade de disponibilidade de uma peça existente. Executa um comando SQL UPDATE e imprime uma mensagem de sucesso se a atualização for bem-sucedida ou uma mensagem de erro caso contrário.
- **atualizarNomePeca(String idPeca, String nome)**: Atualiza o nome de uma peça existente. Executa um comando SQL UPDATE e imprime uma mensagem de sucesso se a atualização for bem-sucedida ou uma mensagem de erro caso contrário.
- **atualizarPrecoPeca(String idPeca, double preco)**: Atualiza o preço de uma peça existente. Executa um comando SQL UPDATE e imprime uma mensagem de sucesso se a atualização for bem-sucedida ou uma mensagem de erro caso contrário.
- **deletar(String id)**: Remove uma peça com o ID especificado da tabela peca. Executa um comando SQL DELETE e imprime uma mensagem de sucesso se a remoção for bem-sucedida ou uma mensagem de erro caso contrário.

Package **servico:**

Este pacote é responsável pela gestão de serviços no sistema, incluindo operações como inserção, atualização, busca e exclusão de serviços. Contém a interface ServicoDAO e a classe concreta ServicoDaoImpl.

1. Interface ServicoDAO

A interface ServicoDAO define os métodos que devem ser implementados para realizar operações sobre serviços no banco de dados.

Métodos definidos:

- **inserir(Servico s)**: Insere um novo serviço no banco de dados. Recebe um objeto Servico e retorna true se a inserção for bem-sucedida e false caso contrário.
- **listar()**: Lista todos os serviços presentes no banco de dados. Retorna uma lista de objetos Servico.
- **buscarPorId(String id)**: Busca um serviço pelo seu ID. Recebe um ID e retorna um objeto Servico. Se o serviço não for encontrado, retorna null.
- **servicoExiste(String idServico)**: Verifica se um serviço com o ID especificado existe no banco de dados. Recebe um ID e retorna true se o serviço existir e false caso contrário.
- **atualizarTipo(String idServico, String tipo)**: Atualiza o tipo de um serviço existente. Recebe o ID do serviço e o novo tipo.
- **atualizarDescricao(String idServico, String descricao)**: Atualiza a descrição de um serviço existente. Recebe o ID do serviço e a nova descrição.
- **atualizarPreco(String idServico, double preco)**: Atualiza o preço de um serviço existente. Recebe o ID do serviço e o novo preço.
- **atualizarDuracao(String idServico, int duracao)**: Atualiza a duração de um serviço existente. Recebe o ID do serviço e a nova duração.
- **deletar(String id)**: Remove um serviço pelo seu ID. Recebe um ID e realiza a exclusão do serviço correspondente no banco de dados.

2. Classe ServicoDaoImpl

A classe ServicoDaoImpl é a implementação concreta da interface ServicoDAO, realizando operações no banco de dados Oracle usando JDBC.

- **Conexão com o Banco de Dados**:
    - Utiliza OracleDataSource para estabelecer uma conexão com o banco de dados Oracle. As credenciais de conexão são fornecidas pela classe Credenciais.
- **Métodos**:
    - **inserir(Servico s)**: Adiciona um novo serviço à tabela servico. Utiliza um comando SQL INSERT e retorna true se a operação for bem-sucedida, e false em caso de erro.

- **listar()**: Recupera todos os serviços da tabela servico. Executa um comando SQL SELECT, cria objetos Servico com os dados retornados e os adiciona a uma lista. Retorna essa lista.
- **buscarPorId(String id)**: Recupera um serviço específico com base no seu ID. Executa um comando SQL SELECT e cria um objeto Servico com os dados retornados. Retorna o objeto ou null se não encontrar o serviço.
- **servicoExiste(String idServico)**: Verifica se um serviço com o ID especificado existe na tabela servico. Executa um comando SQL SELECT COUNT(*) e retorna true se o serviço existir e false caso contrário.
- **atualizarTipo(String idServico, String tipo)**: Atualiza o tipo de um serviço existente. Executa um comando SQL UPDATE e imprime uma mensagem de sucesso se a atualização for bem-sucedida ou uma mensagem de erro caso contrário.
- **atualizarDescricao(String idServico, String descricao)**: Atualiza a descrição de um serviço existente. Executa um comando SQL UPDATE e imprime uma mensagem de sucesso se a atualização for bem-sucedida ou uma mensagem de erro caso contrário.
- **atualizarPreco(String idServico, double preco)**: Atualiza o preço de um serviço existente. Executa um comando SQL UPDATE e imprime uma mensagem de sucesso se a atualização for bem-sucedida ou uma mensagem de erro caso contrário.
- **atualizarDuracao(String idServico, int duracao)**: Atualiza a duração de um serviço existente. Executa um comando SQL UPDATE e imprime uma mensagem de sucesso se a atualização for bem-sucedida ou uma mensagem de erro caso contrário.
- **deletar(String id)**: Remove um serviço com o ID especificado da tabela servico. Executa um comando SQL DELETE e imprime uma mensagem de sucesso se a remoção for bem-sucedida ou uma mensagem de erro caso contrário.

Package **usuario:**

Este pacote é responsável pela gestão de usuários no sistema, incluindo operações como inserção, atualização, busca e exclusão de usuários. Contém a interface UsuarioDAO e a classe concreta UsuarioDaoImpl.

1. Interface UsuarioDAO

A interface UsuarioDAO define os métodos que devem ser implementados para realizar operações sobre usuários no banco de dados.

Métodos:

- **inserir(Usuario usuario)**: Insere um novo usuário no banco de dados. Recebe um objeto Usuario e retorna true se a inserção for bem-sucedida e false caso contrário.
- **listar()**: Lista todos os usuários presentes no banco de dados. Retorna uma lista de objetos Usuario.
- **buscarPorCPF(String cpf)**: Busca um usuário pelo seu CPF. Recebe um CPF e retorna um objeto Usuario. Se o usuário não for encontrado, retorna null.
- **usuarioExiste(String cpf)**: Verifica se um usuário com o CPF especificado existe no banco de dados. Recebe um CPF e retorna true se o usuário existir e false caso contrário.
- **atualizarNome(String cpf, String nome)**: Atualiza o nome de um usuário existente. Recebe o CPF do usuário e o novo nome.
- **atualizarEmail(String cpf, String email)**: Atualiza o e-mail de um usuário existente. Recebe o CPF do usuário e o novo e-mail.
- **atualizarTelefone(String cpf, String telefone)**: Atualiza o telefone de um usuário existente. Recebe o CPF do usuário e o novo telefone.
- **atualizarSenha(String cpf, String senha)**: Atualiza a senha de um usuário existente. Recebe o CPF do usuário e a nova senha.
- **deletar(String cpf)**: Remove um usuário pelo seu CPF. Recebe um CPF e realiza a exclusão do usuário correspondente no banco de dados.
-

2. Classe UsuarioDaoImpl

A classe UsuarioDaoImpl é a implementação concreta da interface UsuarioDAO, realizando operações no banco de dados Oracle usando JDBC.

- **Conexão com o Banco de Dados**:
    - Utiliza OracleDataSource para estabelecer uma conexão com o banco de dados Oracle. As credenciais de conexão são fornecidas pela classe Credenciais.
- **Métodos**:

- **inserir(Usuario usuario)**: Adiciona um novo usuário à tabela usuario. Utiliza um comando SQL INSERT e retorna true se a operação for bem-sucedida, e false em caso de erro.
- **listar()**: Recupera todos os usuários da tabela usuario. Executa um comando SQL SELECT, cria objetos Usuario com os dados retornados e os adiciona a uma lista. Retorna essa lista.
- **buscarPorCPF(String cpf)**: Recupera um usuário específico com base no seu CPF. Executa um comando SQL SELECT e cria um objeto Usuario com os dados retornados. Retorna o objeto ou null se não encontrar o usuário.
- **usuarioExiste(String cpf)**: Verifica se um usuário com o CPF especificado existe na tabela usuario. Executa um comando SQL SELECT COUNT(*) e retorna true se o usuário existir e false caso contrário.
- **atualizarNome(String cpf, String nome)**: Atualiza o nome de um usuário existente. Executa um comando SQL UPDATE e imprime uma mensagem de sucesso se a atualização for bem-sucedida ou uma mensagem de erro caso contrário.
- **atualizarEmail(String cpf, String email)**: Atualiza o e-mail de um usuário existente. Executa um comando SQL UPDATE e imprime uma mensagem de sucesso se a atualização for bem-sucedida ou uma mensagem de erro caso contrário.
- **atualizarTelefone(String cpf, String telefone)**: Atualiza o telefone de um usuário existente. Executa um comando SQL UPDATE e imprime uma mensagem de sucesso se a atualização for bem-sucedida ou uma mensagem de erro caso contrário.
- **atualizarSenha(String cpf, String senha)**: Atualiza a senha de um usuário existente. Executa um comando SQL UPDATE e imprime uma mensagem de sucesso se a atualização for bem-sucedida ou uma mensagem de erro caso contrário.
- **deletar(String cpf)**: Remove um usuário com o CPF especificado da tabela usuario. Executa um comando SQL DELETE e imprime uma mensagem de sucesso se a remoção for bem-sucedida ou uma mensagem de erro caso contrário.

Package **veiculo:**

Este pacote é responsável pela gestão de veículos no sistema, incluindo operações como inserção, atualização, busca e exclusão de veículos. Contém a interface VeiculoDAO e a classe concreta VeiculoDaoImpl.

1. Interface VeiculoDAO

A interface VeiculoDAO define os métodos que devem ser implementados para realizar operações sobre veículos no banco de dados.

Métodos:

- **inserir(Veiculo veiculo)**: Insere um novo veículo no banco de dados. Recebe um objeto Veiculo e retorna true se a inserção for bem-sucedida e false caso contrário.
- **listar()**: Lista todos os veículos presentes no banco de dados. Retorna uma lista de objetos Veiculo.
- **buscarPorPlaca(String placa)**: Busca um veículo pelo seu número de placa. Recebe uma placa e retorna um objeto Veiculo. Se o veículo não for encontrado, retorna null.
- **veiculoExiste(String placa)**: Verifica se um veículo com a placa especificada existe no banco de dados. Recebe uma placa e retorna true se o veículo existir e false caso contrário.
- **atualizarMarca(String placa, String marca)**: Atualiza a marca de um veículo existente. Recebe a placa do veículo e a nova marca.
- **atualizarModelo(String placa, String modelo)**: Atualiza o modelo de um veículo existente. Recebe a placa do veículo e o novo modelo.
- **atualizarAno(String placa, int ano)**: Atualiza o ano de fabricação de um veículo existente. Recebe a placa do veículo e o novo ano.
- **atualizarQuilometragem(String placa, double quilometragem)**: Atualiza a quilometragem de um veículo existente. Recebe a placa do veículo e a nova quilometragem.
- **atualizarCpfProprietario(String placa, String cpf)**: Atualiza o CPF do proprietário de um veículo existente. Recebe a placa do veículo e o novo CPF do proprietário.
- **deletar(String placa)**: Remove um veículo pelo seu número de placa. Recebe uma placa e realiza a exclusão do veículo correspondente no banco de dados.

2. Classe VeiculoDaoImpl

A classe VeiculoDaoImpl é a implementação concreta da interface VeiculoDAO, realizando operações no banco de dados Oracle usando JDBC.

- **Conexão com o Banco de Dados**:
    - Utiliza OracleDataSource para estabelecer uma conexão com o banco de dados Oracle. As credenciais de conexão são fornecidas pela classe Credenciais.
- **Métodos**:
    - **inserir(Veiculo veiculo)**: Adiciona um novo veículo à tabela veiculo. Executa um comando SQL INSERT e retorna true se a operação for bem-sucedida, e false em caso de erro.
    - **listar()**: Recupera todos os veículos da tabela veiculo. Executa um comando SQL SELECT, cria objetos Veiculo com os dados retornados e os adiciona a uma lista. Retorna essa lista.
    - **buscarPorPlaca(String placa)**: Recupera um veículo específico com base na placa. Executa um comando SQL SELECT e cria um objeto Veiculo com os dados retornados. Retorna o objeto ou null se não encontrar o veículo.
    - **veiculoExiste(String placa)**: Verifica se um veículo com a placa especificada existe na tabela veiculo. Executa um comando SQL SELECT COUNT(*) e retorna true se o veículo existir e false caso contrário.
    - **atualizarMarca(String placa, String marca)**: Atualiza a marca de um veículo existente. Executa um comando SQL UPDATE e imprime uma mensagem de sucesso se a atualização for bem-sucedida ou uma mensagem de erro caso contrário.
    - **atualizarModelo(String placa, String modelo)**: Atualiza o modelo de um veículo existente. Executa um comando SQL UPDATE e imprime uma mensagem de sucesso se a atualização for bem-sucedida ou uma mensagem de erro caso contrário.
    - **atualizarAno(String placa, int ano)**: Atualiza o ano de um veículo existente. Executa um comando SQL UPDATE e imprime uma mensagem de sucesso se a atualização for bem-sucedida ou uma mensagem de erro caso contrário.
    - **atualizarQuilometragem(String placa, double quilometragem)**: Atualiza a quilometragem de um veículo existente. Executa um comando SQL UPDATE e imprime uma mensagem de sucesso se a atualização for bem-sucedida ou uma mensagem de erro caso contrário.
    - **atualizarCpfProprietario(String placa, String cpf)**: Atualiza o CPF do proprietário de um veículo existente. Verifica se a placa do veículo e o CPF do

usuário existem antes de executar a atualização. Executa um comando SQL UPDATE e imprime uma mensagem de sucesso se a atualização for bem-sucedida ou uma mensagem de erro caso contrário.

- **deletar(String placa)**: Remove um veículo com a placa especificada da tabela veiculo. Executa um comando SQL DELETE e imprime uma mensagem de sucesso se a remoção for bem-sucedida ou uma mensagem de erro caso contrário.

3 – Procedimentos para Rodar a Aplicação

1. Configuração do Ambiente:

- Certifique-se de que você tenha as seguintes ferramentas instaladas:
- JDK (Java Development Kit) para compilar e rodar o código Java.
- Oracle Database ou outro banco de dados compatível para armazenamento de dados.
- IDE como Eclipse, IntelliJ IDEA ou NetBeans.
- Bibliotecas JDBC em JAR para comunicação com o banco de dados Oracle. Ex: Project --> Properties --> Java Build Path --> Libraries --> Classpath --> Add External JARs... --> Selecione o JDBC --> Apply and Close.

2. Banco de Dados (Oracle):

- Crie as tabelas ou abra o sql no banco de dados usando SQL (presente no arquivo).
- Exemplo de criação da tabela de usuários:

CREATE TABLE usuário (

cpf_usuario CHAR(11) CONSTRAINT usuario_cpf_pk PRIMARY KEY,
nome_usuario VARCHAR(80) CONSTRAINT usuario_nm_nn NOT NULL
CONSTRAINT usuario_nm_unique UNIQUE,

email VARCHAR(255) CONSTRAINT usuario_mail_nn NOT NULL,

telefone CHAR(11) CONSTRAINT usuario_tel_nn NOT NULL,

senha VARCHAR(30) CONSTRAINT usuario_sen_nn NOT NULL, CONSTRAINT chk_senha_usuario CHECK (LENGTH(senha) > 6));

- Configure a conexão com o banco no Java. Utilize OracleDataSource para conectar ao Oracle:

```
OracleDataSource ods = new OracleDataSource();
ods.setURL(URL);
ods.setUser(Credenciais.user);
ods.setPassword(Credenciais.pwd);
return ods.getConnection();
```

- Se deseja utilizar as próprias credenciais para acessar o banco de dados, altere as informações na package ‘credenciais’ dentro de ‘dao’. Caso deseje utilizar as credenciais do aluno, utilize user ‘rm554424’ e pwd ‘040704’.

3. Estrutura do Código (Java):

- A aplicação segue o padrão MVC (Model-View-Controller):
- Model: Contém classes para as entidades (Usuário, Veículo, Serviço etc.) e a lógica de negócios.
- View: Interação com o usuário via menus, formulários, ou interface gráfica.
- Controller: Intermedia a interação entre a View e o Model, controlando as operações e lógica.
- **Compilação:**

- Se estiver utilizando uma IDE, basta configurar o projeto e rodar a partir da IDE.

4. Operações CRUD (DAO):

- As classes DAO são responsáveis pela interação com o banco de dados. Implemente as operações CRUD:
- Inserção: Adicione novos dados ao banco (ex.: novos usuários, veículos etc.).
- Leitura: Consulte dados existentes (ex.: lista de serviços, diagnósticos etc.).

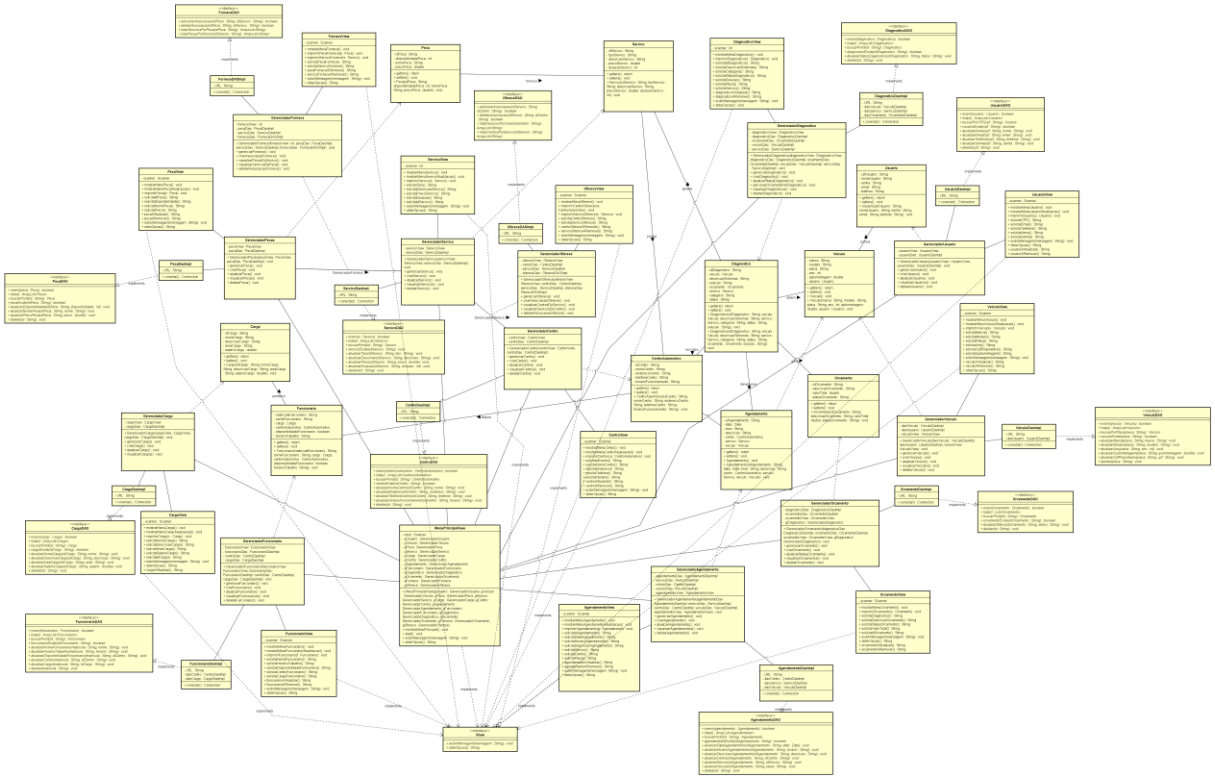
- Atualização: Atualize informações no banco (ex.: atualizar status de serviços).
- Exclusão: Remova dados conforme necessário.
- Exemplo de inserção de usuário em Java:

```
public boolean inserir(Usuario usuario) {
   String sql = "INSERT INTO usuario (cpf_usuario, nome_usuario, email,
telefone, senha) VALUES (?,?,?,?,?)";
   try (Connection conn = conectar();
        PreparedStatement ps = conn.prepareStatement(sql)) {
      ps.setString(1, usuario.getCpfUsuario());
      ps.setString(2, usuario.getNomeUsuario());
      ps.setString(3, usuario.getEmail());
      ps.setString(4, usuario.getTelefone());
      ps.setString(5, usuario.getSenha());
      int rowsAffected = ps.executeUpdate();
      return rowsAffected > 0;
   } catch (SQLException e) {
      System.err.println("Ocorreu um erro ao inserir usuário:");
      e.printStackTrace();
      return false;
   }
}
```

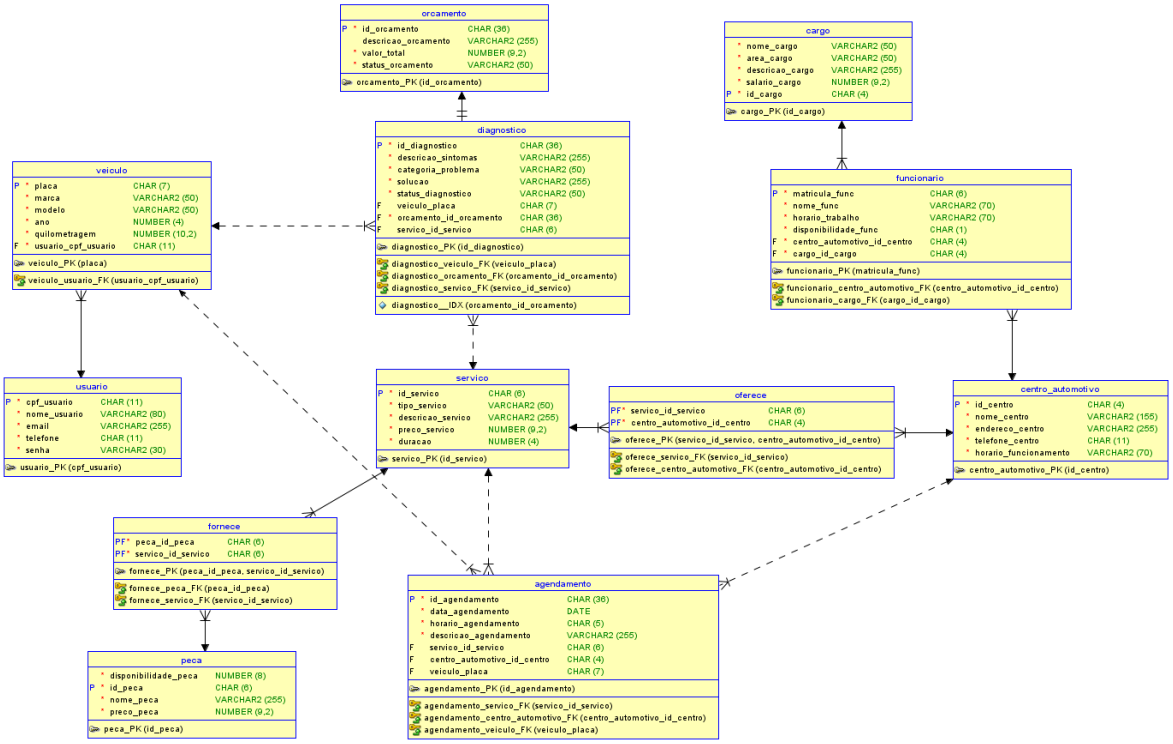
5. Testes:

- Execute a aplicação e teste cada funcionalidade, como:
- Cadastro e login de usuários.
- Inserção e consulta de veículos, diagnósticos, agendamentos.
- Verifique a integração com o banco de dados.

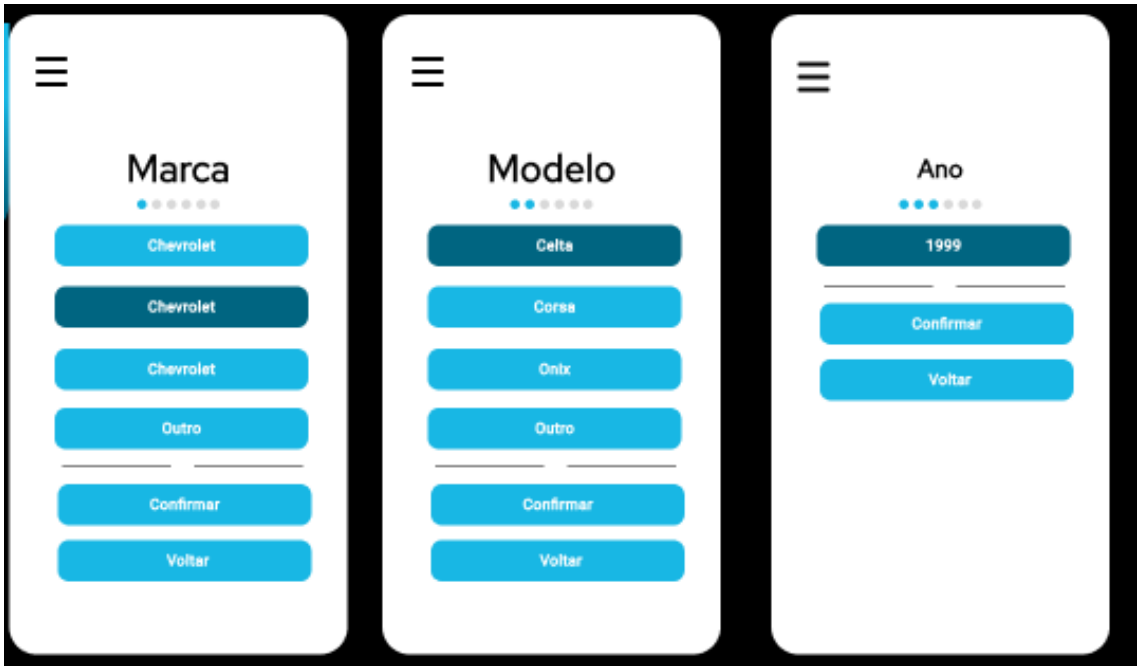
4 – Diagrama de Classes



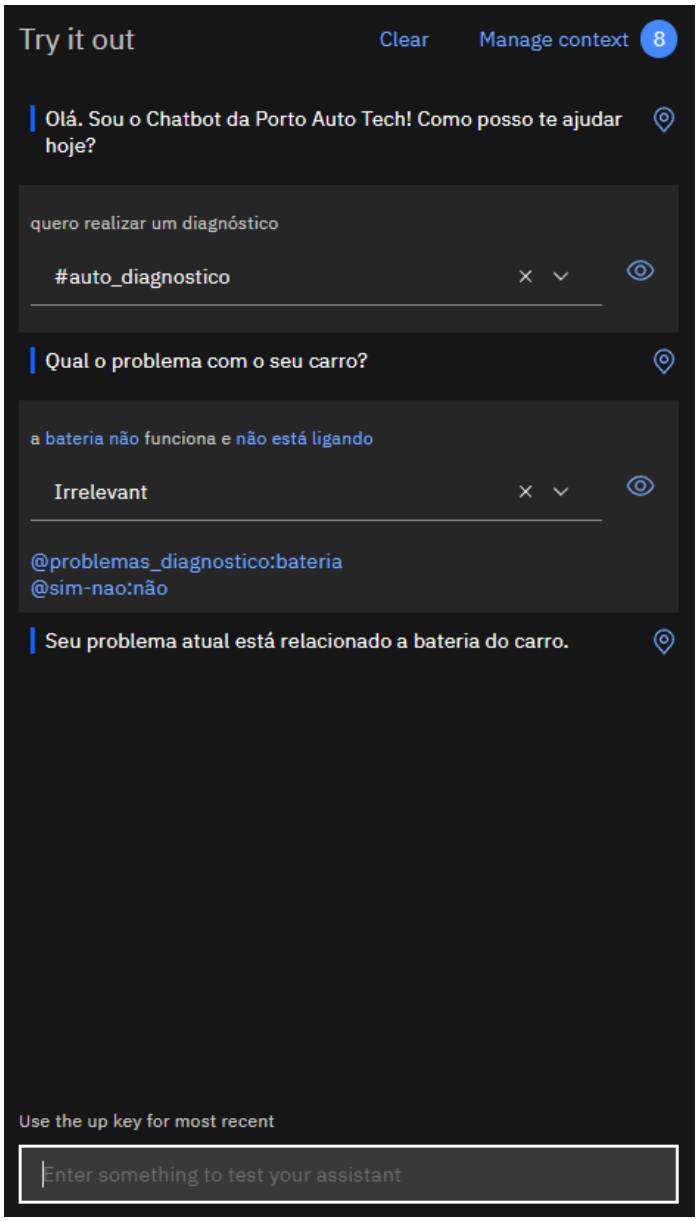
5 – MER


orcamento
P * id_orcamento CHAR (36)
descricao_orcamento VARCHAR2 (255)
* valor_total NUMBER (9,2)
* status_orcamento VARCHAR2 (50)
orcamento_PK (id_orcamento)
cargo
* nome_cargo VARCHAR2 (50)
* area_cargo VARCHAR2 (50)
* descricao_cargo VARCHAR2 (255)
* salario_cargo NUMBER (9,2)
P * id_cargo CHAR (4)
cargo_PK (id_cargo)
diagnostico
P * id_diagnostico CHAR (36)
* descricao_sintomas VARCHAR2 (255)
* categoria_problema VARCHAR2 (50)
* solucao VARCHAR2 (255)
* status_diagnostico VARCHAR2 (50)
F veiculo_placa CHAR (7)
F * orcamento_id_orcamento CHAR (36)
F servico_id_servico CHAR (6)
diagnostico_PK (id_diagnostico)
diagnostico_veiculo_FK (veiculo_placa)
diagnostico_orcamento_FK (orcamento_id_orcamento)
diagnostico_servico_FK (servico_id_servico)
diagnostico__IDX (orcamento_id_orcamento)
veiculo
P * placa CHAR (7)
* marca VARCHAR2 (50)
* modelo VARCHAR2 (50)
* ano NUMBER (4)
* quilometragem NUMBER (10,2)
F * usuario_cpf_usuario CHAR (11)
veiculo_PK (placa)
veiculo_usuario_FK (usuario_cpf_usuario)
funcionario
P * matricula_func CHAR (6)
* nome_func VARCHAR2 (70)
* horario_trabalho VARCHAR2 (70)
* disponibilidade_func CHAR (1)
F * centro_automotivo_id_centro CHAR (4)
F * cargo_id_cargo CHAR (4)
funcionario_PK (matricula_func)
funcionario_centro_automotivo_FK (centro_automotivo_id_centro)
funcionario_cargo_FK (cargo_id_cargo)
usuario
P * cpf_usuario CHAR (11)
* nome_usuario VARCHAR2 (80)
* email VARCHAR2 (255)
* telefone CHAR (11)
* senha VARCHAR2 (30)
usuario_PK (cpf_usuario)
servico
P * id_servico CHAR (6)
* tipo_servico VARCHAR2 (50)
* descricao_servico VARCHAR2 (255)
* preco_servico NUMBER (9,2)
* duracao NUMBER (4)
servico_PK (id_servico)
oferece
PF * servico_id_servico CHAR (6)
PF * centro_automotivo_id_centro CHAR (4)
oferece_PK (servico_id_servico, centro_automotivo_id_centro)
oferece_servico_FK (servico_id_servico)
oferece_centro_automotivo_FK (centro_automotivo_id_centro)
centro_automotivo
P * id_centro CHAR (4)
* nome_centro VARCHAR2 (155)
* endereco_centro VARCHAR2 (255)
* telefone_centro CHAR (11)
* horario_funcionamento VARCHAR2 (70)
centro_automotivo_PK (id_centro)
fornece
PF * peca_id_peca CHAR (6)
PF * servico_id_servico CHAR (6)
fornece_PK (peca_id_peca, servico_id_servico)
fornece_peca_FK (peca_id_peca)
fornece_servico_FK (servico_id_servico)
agendamento
P * id_agendamento CHAR (36)
* data_agendamento DATE
* horario_agendamento CHAR (5)
* descricao_agendamento VARCHAR2 (255)
F servico_id_servico CHAR (6)
F centro_automotivo_id_centro CHAR (4)
F veiculo_placa CHAR (7)
agendamento_PK (id_agendamento)
agendamento_servico_FK (servico_id_servico)
agendamento_centro_automotivo_FK (centro_automotivo_id_centro)
agendamento_veiculo_FK (veiculo_placa)
peca
* disponibilidade_peca NUMBER (8)
P * id_peca CHAR (6)
* nome_peca VARCHAR2 (255)
* preco_peca NUMBER (9,2)
peca_PK (id_peca)

6 – Exemplo de protótipo



O usuário seleciona cada opção de cadastro do carro (marca, modelo, ano, quilometragem e placa), até que possa cadastrá-lo. Com isso, o veículo será inserido no banco de dados.



Através do Chatbot, podemos perguntar ao usuário qual o problema que ele está tendo com o carro. Ao obter o problema identificado, podemos gerar o diagnóstico e inseri-lo no banco de dados.