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 2

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;

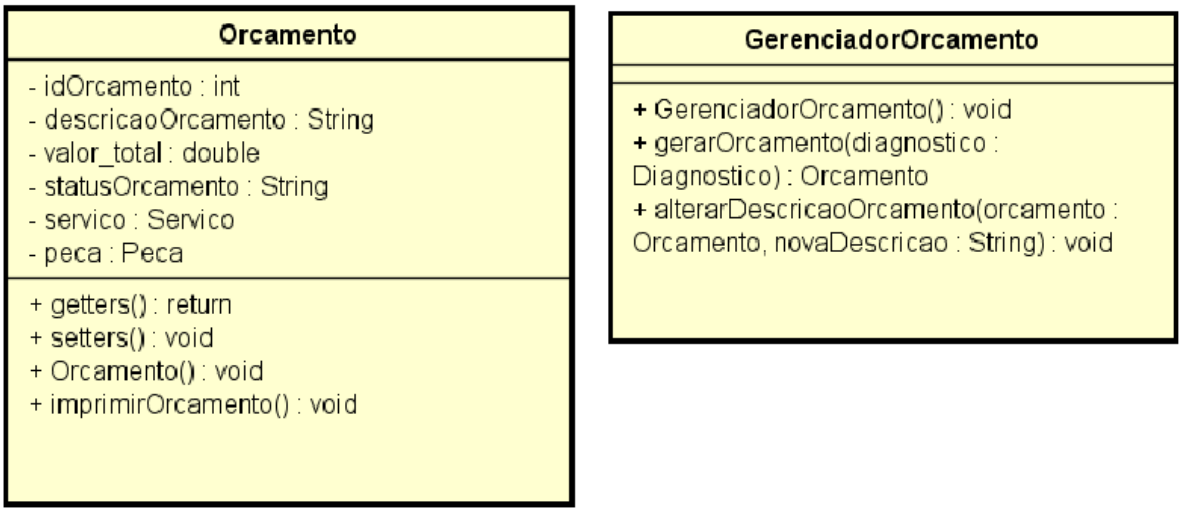
- comunicação através de voz (speech-to-text) com o Chatbot, permitindo descrever o problema oralmente ou enviar ruídos emitidos pelo carro, que serão analisados pela I.A;

- ligação em chamada com mecânico através do Chatbot, explicando o progresso da manutenção.

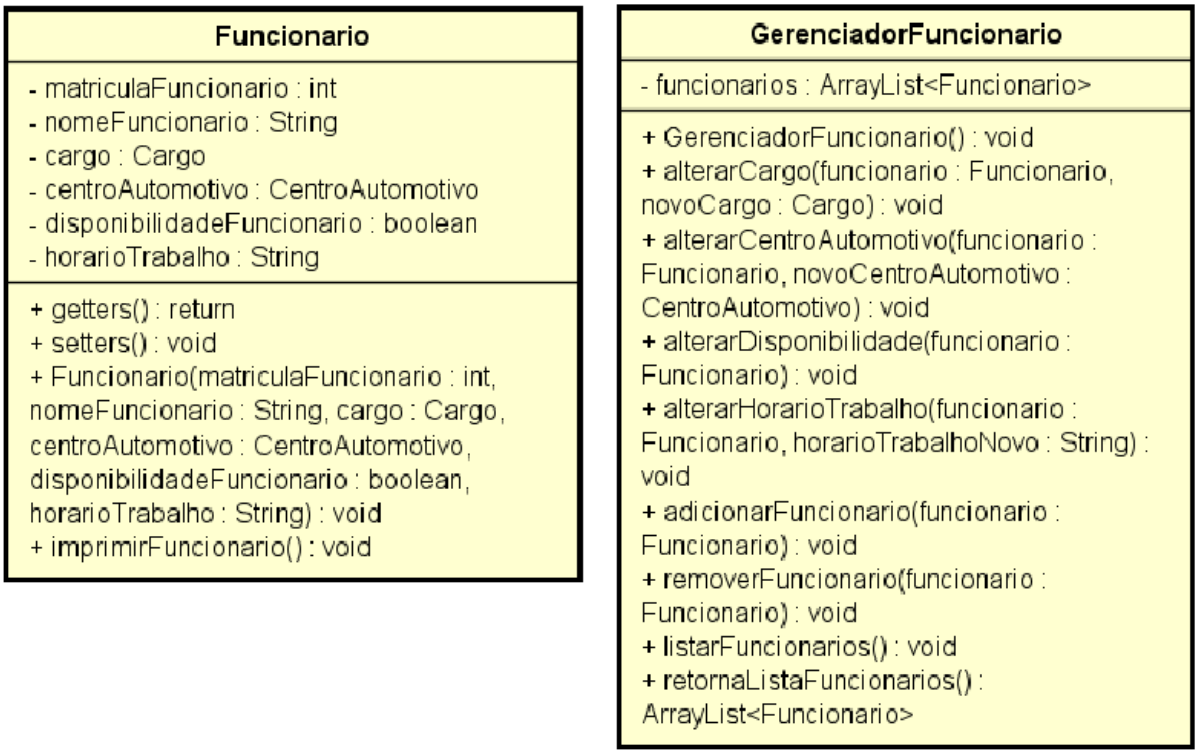
Através das funcionalidades descritas, o sistema será capaz de auxiliar pessoas que enfrentam dificuldades no entendimento dos problemas do veículo, além de fornecer diferenciais que auxiliarão em necessidades dos clientes que poucas empresas oferecem.

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.

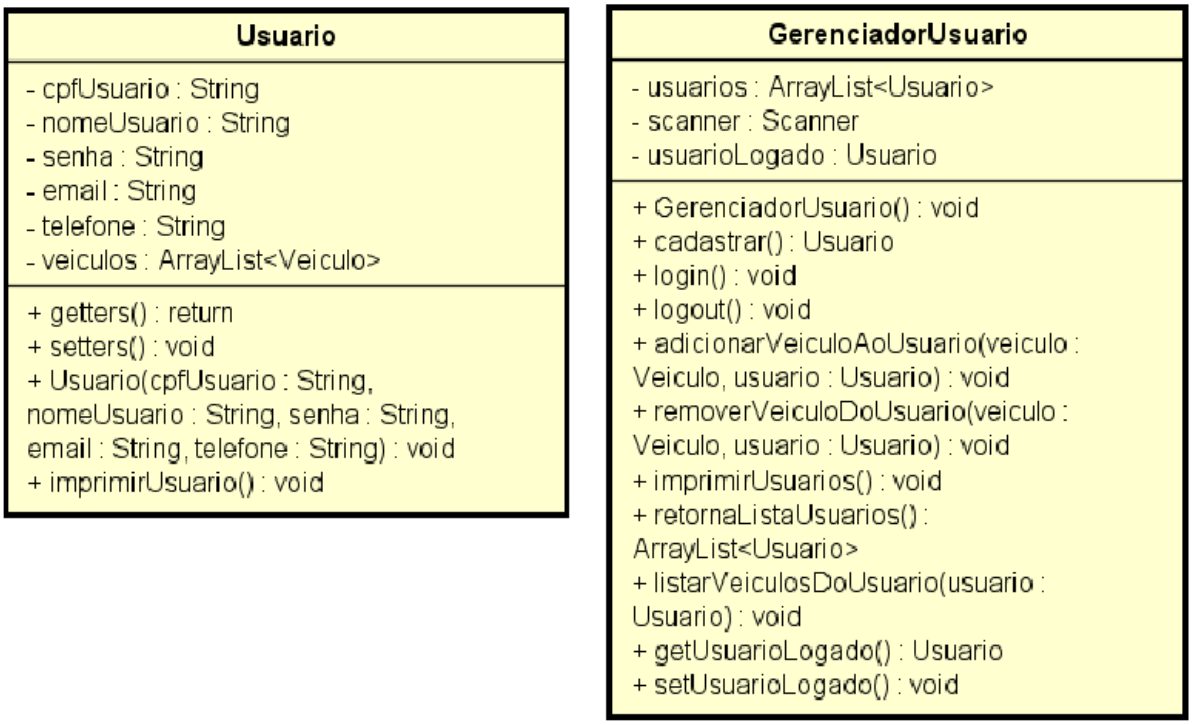
2 – Diagrama de Classes UML



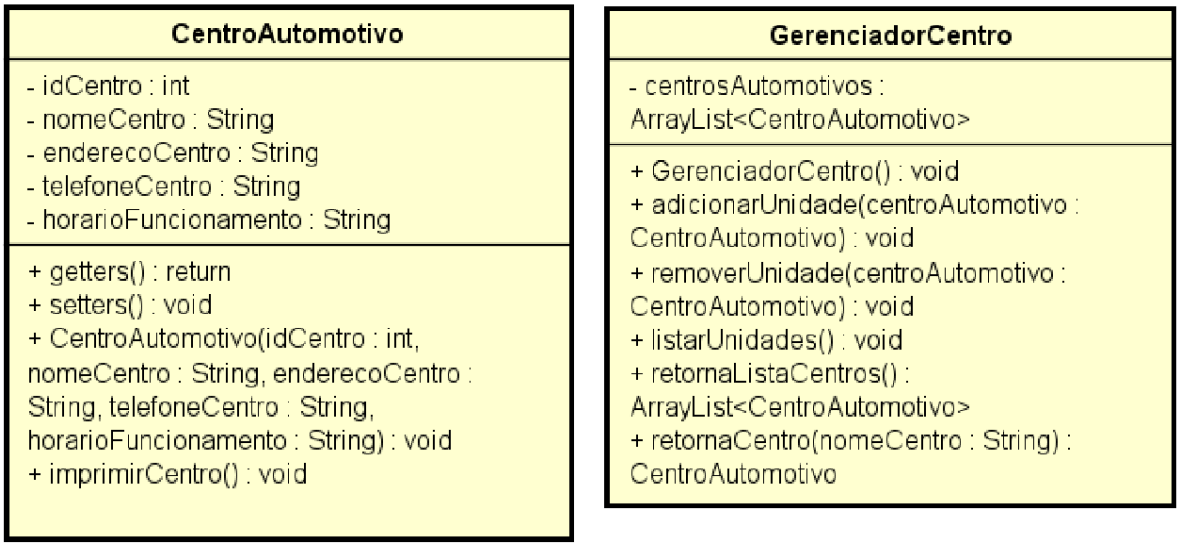
- **Orcamento:** classe Model que armazena informações sobre um orçamento (id, descrição, valor total, status, serviço e peça). Possui métodos getters e setters, construtor vazio e um método para imprimir os dados do orçamento;
- **GerenciadorOrcamento:** gerenciador da classe Orcamento, responsável por gerar um novo orçamento com base no diagnóstico;



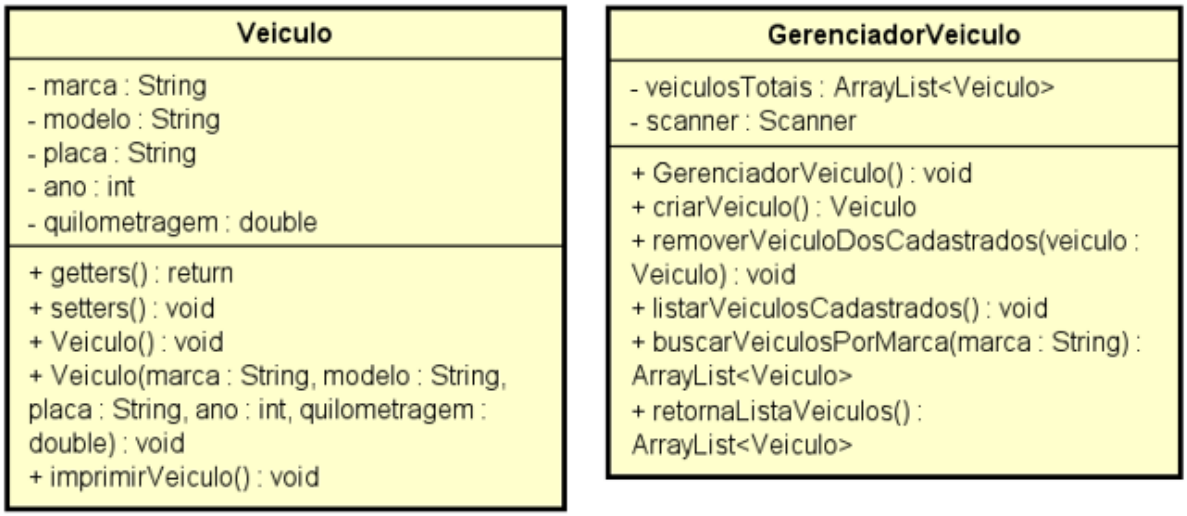
- **Funcionario:** classe Model que armazena informações sobre um funcionário (matrícula, nome, cargo, centro automotivo, disponibilidade e horário de trabalho). Possui métodos getters e setters, construtor parametrizado e um método para imprimir os dados do funcionário;
- **GerenciadorFuncionario:** gerenciador de Funcionario, possui um ArrayList de funcionários como atributo. Pode alterar o cargo, centro automotivo, disponibilidade e horário de trabalho do funcionário. Também pode adicionar e remover o funcionário da lista de funcionários, além de imprimir e retornar a lista.



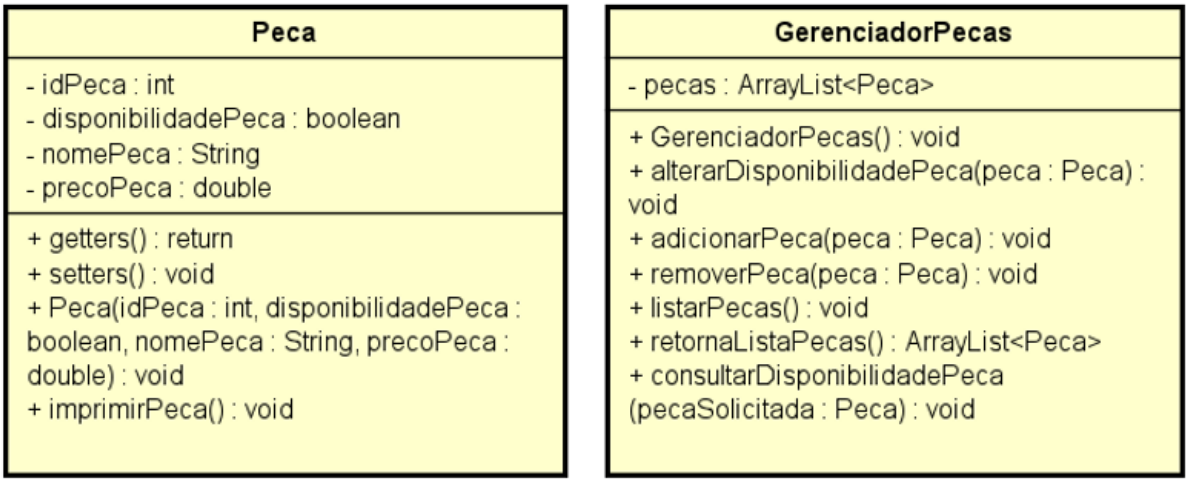
- **Usuario:** classe model que armazena informações sobre um usuário (CPF, nome, senha, e-mail, telefone e uma lista de veículos. Possui métodos getters e setters, um construtor parametrizado e um método para imprimir os dados do usuário;
- **GerenciadorUsuario:** gerenciador de Usuario, possui um ArrayList de usuários, o Scanner e um usuarioLogado com getters e setters.
    - cadastra um usuário, pedindo cpf, nome, senha, telefone e e-mail através de inputs.
    - loga o usuário, solicitando e-mail e senha. Caso os dados fornecidos coincidam com algum usuário cadastrado, autoriza o acesso. O atributo usuarioLogado é atualizado com os dados do usuário que acabou de entrar no sistema;
    - quando o usuário faz logout, o usuarioLogado é definido como null, permitindo um novo login;
    - adiciona e remove veículos da lista do usuário logado, além imprimir e retornar a lista de usuários, e imprimir a lista de veículos do usuário;



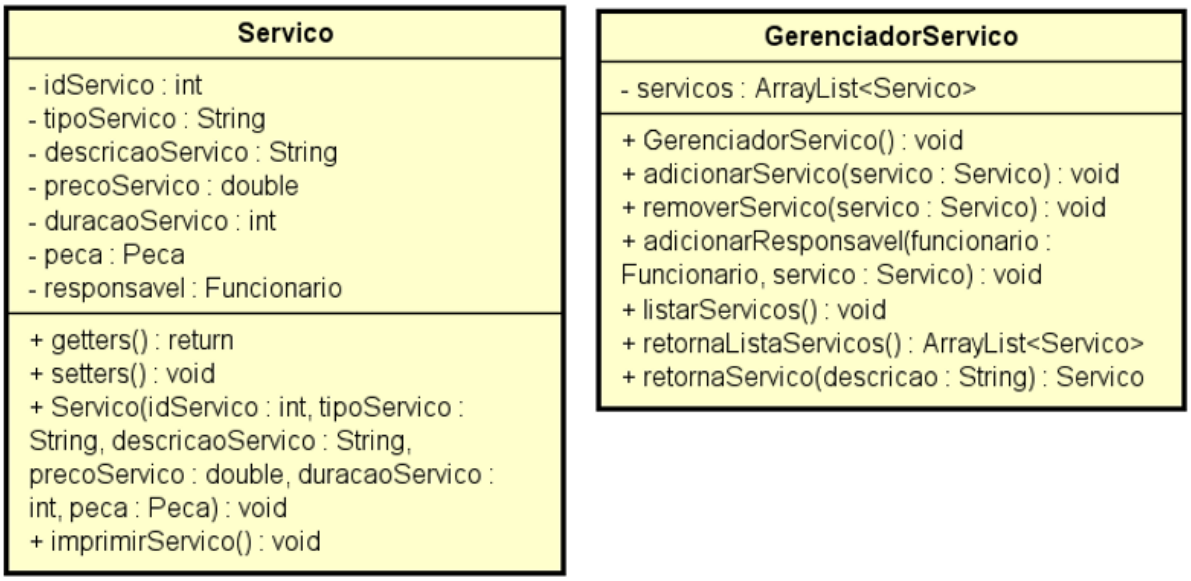
- **CentroAutomotivo:** classe Model que armazena informações sobre o centro automotivo (id, nome, endereço, telefone e horário de funcionamento). Possui métodos getters e setters, construtor parametrizado e um método para imprimir os dados do centro;
- **GerenciadorCentro:** gerenciador de CentroAutomotivo, possui um ArrayList de CentroAutomotivo como atributo. Pode adicionar e remover centros na lista, imprimir e retornar a lista, além de retornar um objeto CentroAutomotivo baseado em seu nome;



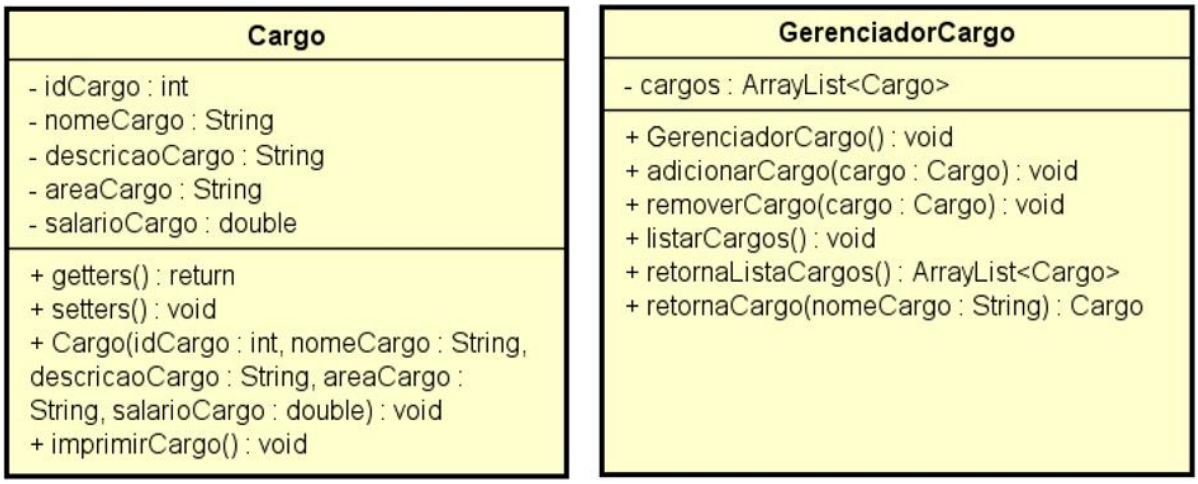
- **Veiculo:** classe Model que armazena informações sobre um veículo (marca, modelo, placa, ano e quilometragem). Possui métodos getters e setters, construtor parametrizado e um método para imprimir os dados do veículo;
- **GerenciadorVeiculo:** gerenciador de Veiculo, possui um ArrayList de Veiculo e um Scanner como atributos. Pode criar um veículo, pedindo marca, modelo, placa, ano e quilometragem ao usuário através de inputs.



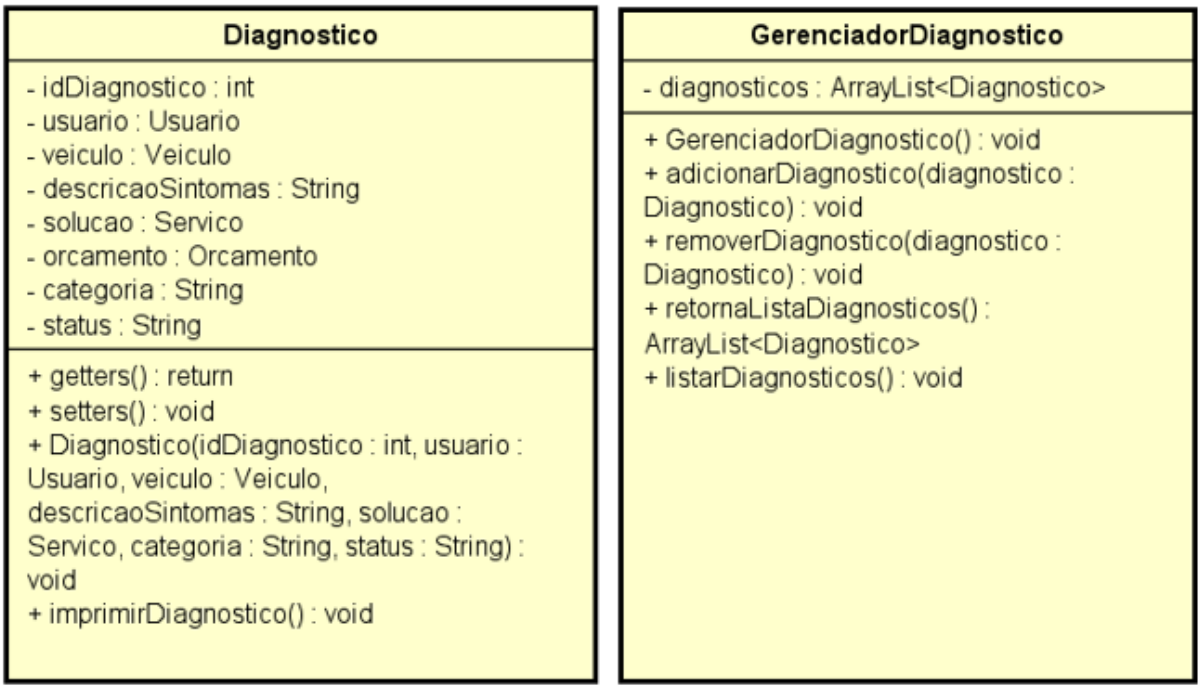
- **Peca:** classe Model que armazena informações de uma peça (id, disponibilidade, nome e preço). Possui métodos getters e setters, construtor parametrizado e um método para imprimir os dados da peça;
- **GerenciadorPecas:** gerenciador de Peca, possui um ArrayList de Peca como atributo. Ele é responsável por alterar e consultar a disponibilidade da peça, adicionar ou remover uma peça da lista de peças e imprimir e retornar a lista;



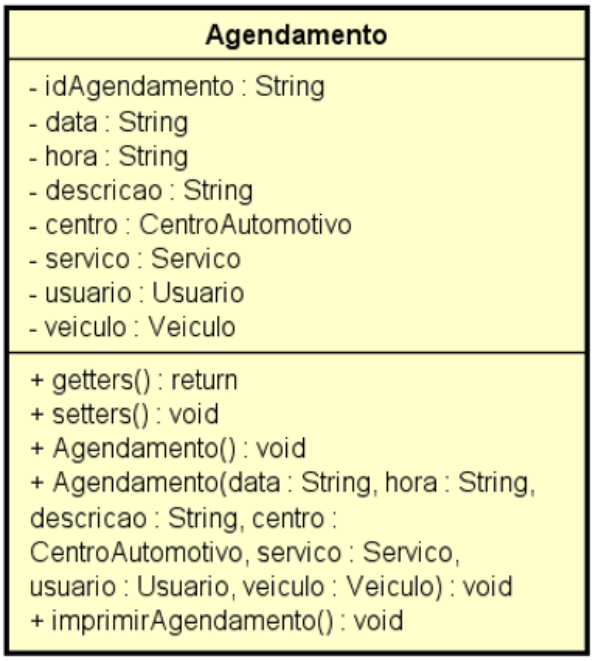
- **Servico:** classe Model que armazena informações de um serviço (id, tipo, descrição, preço, duração, peça e responsável). Possui métodos getters e setters, construtor parametrizado e um método que imprime os dados do serviço;
- **GerenciadorServico:** gerenciador de Servico, possui um ArrayList de Servico como atributo. Ele é responsável por adicionar e remover um serviço da lista, pode adicionar um responsável (classe Funcionario) a um serviço, pode imprimir e retornar a lista de serviços, além de retornar um objeto Servico baseado na descrição;

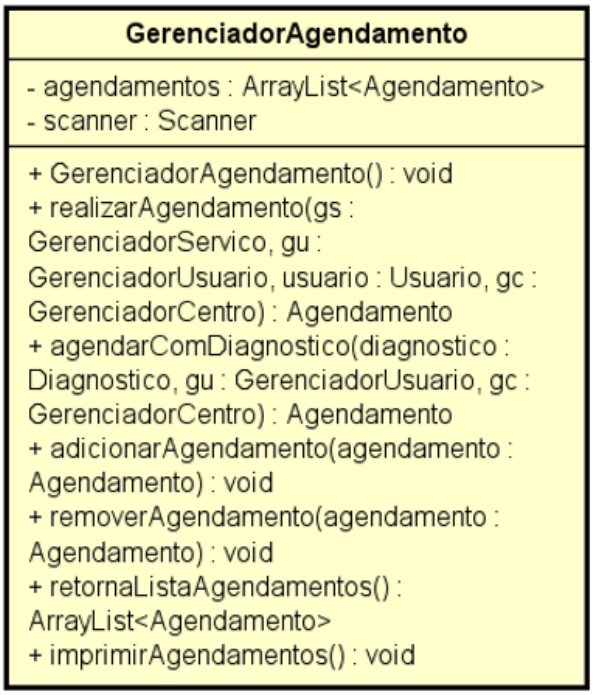


- **Cargo:** classe Model que armazena informações de um cargo (id, nome, descrição, área e salário). Possui métodos getters e setters, construtor parametrizado e um método para imprimir os dados do cargo;
- **GerenciadorCargo:** gerenciador de Cargo, possui um ArrayList de Cargo como atributo. Ele é responsável por adicionar e remover um cargo da lista, imprimir e retornar a lista de cargos, além de retornar um objeto Cargo baseado em seu nome;



- **Diagnostico:** classe Model que armazena informações de um diagnóstico (id, usuário, veículo, descrição dos sintomas, solução, orçamento, categoria e status). Possui métodos getters e setters, construtor parametrizado e um método para imprimir os dados do diagnóstico;
- **GerenciadorDiagnostico:** gerenciador de Diagnostico, possui um ArrayList de Diagnostico como atributo. Ele é responsável por adicionar e remover diagnósticos da lista, além de imprimir e retornar a lista de diagnósticos.





- **Agendamento:** classe Model que armazena informações de um agendamento (id, data, hora, descrição, centro automotivo, serviço, usuário e veículo). Possui métodos getters e setters, construtor vazio, construtor parametrizado e um método para imprimir os dados do agendamento;
- **GerenciadorAgendamento:** gerenciador de Agendamento, possui um ArrayList de Agendamento e um Scanner como atributos. Ele é responsável por criar um Agendamento, pedindo ao usuário a data, o horário, o serviço, o centro automotivo e o veículo através de inputs. O usuário também pode fazer um agendamento utilizando o diagnóstico obtido, precisando apenas informar a data, o horário e o centro automotivo. O gerenciador também pode adicionar e remover um agendamento da lista, além de imprimir e retornar a lista de agendamentos;