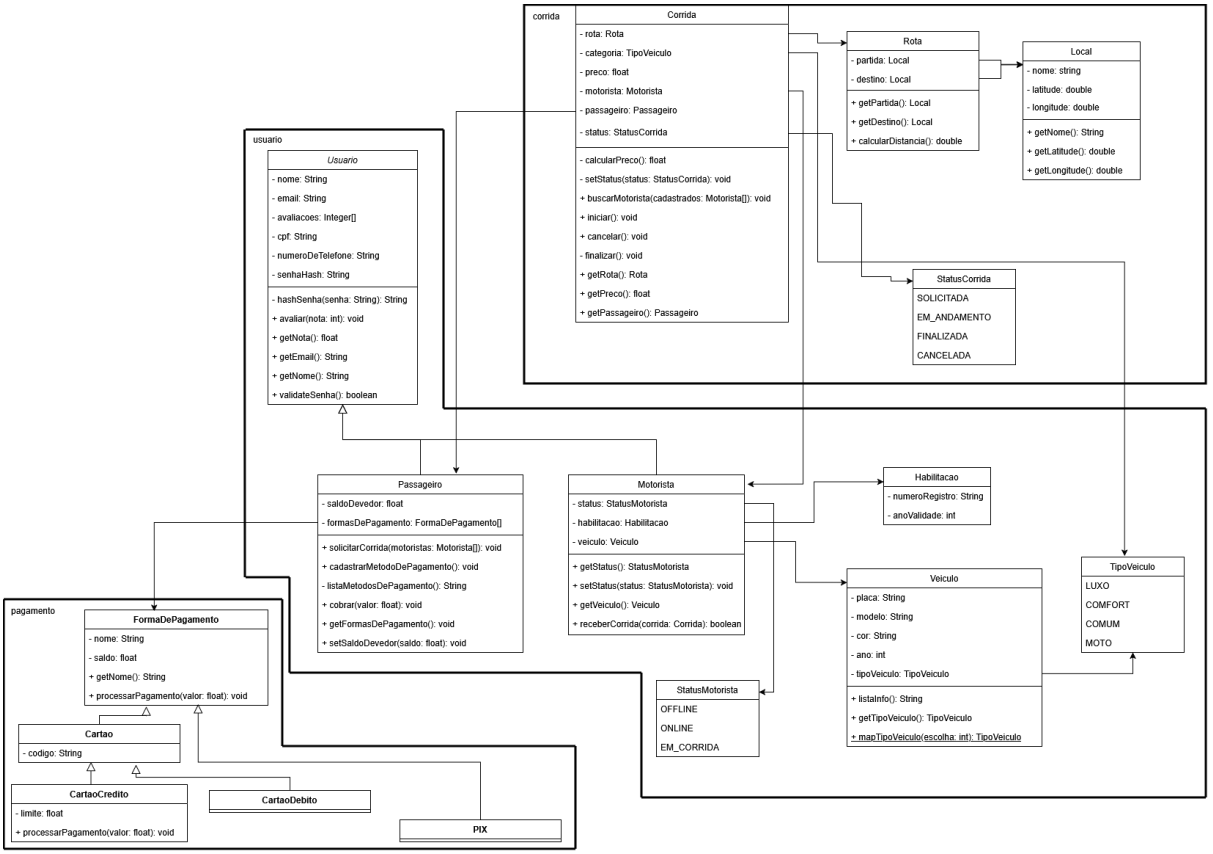
Projeto OO - André Lanna

Data de entrega: 03/12/2025

Componentes: Eliabe (*251020208*), Gabriel Melo (*251009185*), Igor (*251010186*)

Diagrama de classes



Estrutura do projeto

Heranças

- Iniciamos o projeto focando na entidade principal (**Corrida**) e nos dois principais agentes com os quais ela interage, que são **Passageiro** e **Motorista**. Como elas compartilham de quase todos os atributos, ter uma classe abstrata de onde ambos herdam é uma escolha obrigatória para este caso. Por isso, criamos e decidimos nomeá-la de **Usuario**.
- Outras duas relações de herança que decidimos criar foram entre **Cartao** e **PIX** com a sua classe abstrata **FormaDePagamento** (por todas serem

formas de pagamento) e mais um nível abaixo no caso dos cartões, onde **CartaoCredito** e **CartaoDebito** herdarão a classe abstrata **Cartao**.

Associações

O sistema do aplicativo utiliza muito das associações, arrisco a dizer que temos mais atributos associados a outros objetos do que a tipos primitivos.

Segue a lista das associações criadas:

- Corrida → Rota (1:1): Cada corrida possui uma rota definida;
- Corrida → Motorista (1:1): Cada corrida possui um motorista;
- Corrida → Passageiro (1:1): Cada corrida possui um passageiro que a solicitou e pagará;
- Rota → Local (1:2): Uma rota possui dois locais referenciados, um de partida e outro de destino;
- Passageiro → FormaDePagamento (1:N): Um passageiro pode cadastrar quantas formas de pagamento quiser;
- Motorista → Habilitação (1:1): Cada motorista possui sua CNH;
- Motorista → Veículo (1:1): Cada motorista possui seu veículo;

Polimorfismos

Os polimorfismos nos ajudaram muito no trabalho. Eles facilitaram a criação das nossas interfaces e deixaram o código fácil de entender.

Por sobrecarga

Acredito que este foi o mais útil, utilizamos ele em diversos métodos construtores, para conseguir instanciar objetos tanto diretamente pelo código, quanto pela linha de comando. Sem precisar criar métodos estáticos para isso, que iriam reduzir a modularidade do código e possivelmente aumentar a complexidade.

Esta estratégia foi tão útil que a utilizamos nos métodos construtores de Corrida, Motorista, Passageiro, Local, Rota, Habilitacao, Veiculo, CartaoCredito, CartaoDebito e PIX (ou seja, em **todas** as classes, exceto as abstratas).

Por sobrescrita

Utilizamos pouco do polimorfismo por sobrescrita, mas não deixamos de utilizar. Ele foi bastante útil na classe CartaoCredito, que processa o pagamento de uma forma um pouco diferente das demais, por descontar o valor cobrado do limite e não de um saldo.

Por coerção

Utilizamos polimorfismo por coerção nos métodos que tratam a respeito do dinheiro, transformando tudo para float.

Paramétricos

Utilizamos o polimorfismo paramétrico nas listas presentes nas classes. Na Main para listar os usuários cadastrados, no Passageiro para guardar a lista dos métodos de pagamento e na lista de avaliações de cada usuário, como é descrito no diagrama.

Exceções

Utilizamos as 4 exceções listadas no enunciado:

- SaldoInsuficienteException: quando não há saldo suficiente para pagamento
- PagamentoRecusadoException: quando a operadora de cartão nega o pagamento
- NenhumMotoristaDisponivelException: quando não há motorista disponivel
- EstadoInvalidoDaCorridaException: quando a conversão de estado é inválida

Além dessas, criamos também a **MotoristaInvalidoException**, que é lançada caso o usuário tente fazer o cadastro de um motorista com CNH vencida.