

# Sistema de Mobilidade Urbana (Ride-Sharing)

## Explicação das Classes, Associações e Exceções

O pacote **entidades** contém as classes principais do sistema, responsáveis por representar os elementos do domínio da aplicação, como usuários, veículos, corridas e métodos de pagamento. A seguir, cada classe é explicada em detalhes.

**Usuario (abstrata):** Classe base que centraliza os atributos comuns de todos os usuários (nome, CPF, email, telefone, senha). Serve de base para herança por Motorista e Passageiro.

**Passageiro:** Representa o cliente que solicita corridas. Pode cadastrar múltiplos métodos de pagamento e avaliar motoristas. Contém métodos como *solicitarCorrida()* e *avaliarMotorista()*.

**Motorista:** Responsável por atender solicitações de corrida. Possui CNH válida, veículo associado e status de disponibilidade. Pode aceitar, finalizar corridas e avaliar passageiros.

**Veiculo:** Descreve o carro utilizado pelo motorista (placa, modelo, cor, ano). Está associado a apenas um motorista ativo.

**Corrida:** É a entidade central. Representa uma viagem solicitada, contendo informações sobre origem, destino, distância, categoria de serviço e status (solicitada, aceita, em andamento, finalizada). Também associa o motorista, passageiro e método de pagamento.

**MetodoPagamento (abstrata):** Define a interface de processamento de pagamento. Suas subclasses (Cartão, PIX, Dinheiro) implementam o método *processarPagamento()* de forma polimórfica.

**PagamentoCartao:** Simula pagamentos via cartão de crédito, podendo lançar a exceção *PagamentoRecusadoException*.

**PagamentoPIX:** Representa pagamentos via PIX, processados instantaneamente.

**PagamentoDinheiro:** Simula saldo virtual do passageiro, podendo lançar *SaldoInsuficienteException*.

O pacote **servicos** centraliza a lógica do sistema, representada pela classe **ServicoCorrida**, responsável por encontrar motoristas disponíveis, calcular preços e gerenciar o ciclo de vida das corridas. Métodos como *solicitarCorrida()* e *calcularPreco()* fazem parte dessa classe, garantindo a modularidade e separação das regras de negócio.

O pacote **excecoes** contém classes de exceção personalizadas, fundamentais para o tratamento de situações anormais no sistema. Essas exceções aumentam a robustez e evitam falhas inesperadas, garantindo um comportamento controlado em situações de erro.

**SaldoInsuficienteException:** Lançada quando o método de pagamento em dinheiro não possui saldo suficiente. Evita que a corrida finalize incorretamente e marca-a como pendente de pagamento.

**PagamentoRecusadoException:** Ocorre quando a operadora de cartão nega o pagamento. O sistema registra a falha e impede novas solicitações até a regularização.

**NenhumMotoristaDisponivelException:** É disparada quando não há motoristas online no momento da solicitação. Informa o passageiro e cancela o pedido.

**EstadoInvalidoDaCorridaException:** Previne mudanças de estado incoerentes, como tentar finalizar uma corrida que ainda não começou.

**PassageiroPendenteException:** Bloqueia passageiros com pendências financeiras de solicitar novas corridas.

**MotoristaInvalidoException:** Impede motoristas com CNH vencida ou veículo não aprovado de ficarem disponíveis.

As associações entre as classes seguem o comportamento natural do domínio da aplicação:

- Um **Passageiro** pode ter vários **MetodosPagamento** cadastrados (1:N).
- Cada **Motorista** possui exatamente um **Veiculo** ativo (1:1).
- Cada **Corrida** relaciona um **Passageiro**, um **Motorista** e um **MetodoPagamento** (1:1).
- O **ServicoCorrida** gerencia vários **Motoristas** disponíveis (1:N).

Essas associações refletem a realidade operacional do sistema de mobilidade e respeitam os princípios de encapsulamento e modularidade.

Os principais conceitos de orientação a objetos foram aplicados de forma consistente:

- **Herança:** implementada entre Usuario → Passageiro/Motorista e MetodoPagamento → subclasses.
- **Polimorfismo:** uso da sobrescrita do método processarPagamento() nas subclasses de MetodoPagamento.
- **Encapsulamento:** todos os atributos privados com acesso controlado por getters e setters.
- **Generics:** utilização de listas genéricas (List, List).
- **Tratamento de Exceções:** garante estabilidade frente a falhas de pagamento, motoristas inválidos ou mudanças de estado ilegais.

Esses conceitos asseguram clareza, reuso de código e manutenção facilitada.