



**UNIVERSIDADE DE BRASÍLIA - FCTE - GAMA**  
**ORIENTAÇÃO POR OBJETOS - PROF. ANDRÉ LUIZ PERON MARTINS LANNA**

DISCENTES:  
FABIANA REIS SOUZA,  
ISRAEL SOARES DE PAIVA,  
MURILO RAMALHO CRUZ.

---

**TRABALHO PRÁTICO DE ORIENTAÇÃO POR OBJETOS**

Este presente relatório tem como objetivo apresentar nosso projeto do trabalho prático, um **Sistema de Mobilidade Urbana (Ride-Sharing)**. O objetivo deste projeto é modelar e desenvolver um protótipo de sistema para um aplicativo de compartilhamento de corridas, similar a plataformas como Uber, 99 ou Cabify, aplicando conceitos visto durante as aulas de Orientação a Objetos com a linguagem Java.

Nosso projeto foi dividido em 3 pacotes: *Entidades*, *Serviços* e *Exceptions*, além de um pacote *utils* para uma funcionalidade que foi necessário implementar. Nosso projeto possui as seguintes classes, implementadas de acordo com o contexto do sistema:

**1. Descrição das Classes:**

*Entidades*

**CLASSE USUÁRIO:**

A Classe Usuário reúne as características que qualquer usuário, seja motorista ou passageiro, possa ter como: nome, CPF, e-mail, telefone e uma senha, que validam o cadastro dele no aplicativo, além de atributos que coletam feedbacks, como *avaliacoes* e *mediaAvaliacoes*, e o atributo *possuiPendencias*, que impede de realizar certas tarefas no aplicativo. Usuário é uma classe abstrata, que faz parte do pacote de *Entidades*, pois não existe um usuário genérico, ou ele é um motorista ou um passageiro, assim se alguém tentar



instanciar um objeto do tipo Usuário o sistema não irá permitir. Como métodos ela possui um Construtor que guarda dados pessoais e senha, avaliações e inicia pendência como falso, todo usuário inicia sem pendências no aplicativo. Temos um método público para adicionar avaliação e um método privado para atualizar média, esse método é privado pois não queremos que qualquer um atualize a média manualmente ao chamar esse método, é como se ao receber uma nova avaliação esse método atualiza automaticamente com base no que temos no ArrayList de adicionarAvaliacao.

### **CLASSE PASSAGEIRO:**

A classe passageiro é uma subclasse de Usuário, que representa o cliente que solicita o serviço de transporte. Possui além dos atributos herdados de Usuário, atributos específicos como a lista de metodosPagamento e uma corridaAtual. Implementa os métodos solicitarCorrida, que cria um novo objeto do tipo Corrida e verifica se existe algum motorista disponível para a viagem antes de prosseguir, caso não haja é lançado a exceção NenhumMotoristaException, e também o método cancelarCorrida, que altero o estado da Corrida se ela ainda não estiver em andamento.

### **CLASSE MOTORISTA:**

A classe Motorista também é uma subclasse de Usuário, que representa o prestador de serviço de transportes. Além dos atributos herdados do usuário, possui como atributos específicos a disponibilidade, veículo, cnh e uma corridaAtual. Ele implementa métodos de gerenciamento de corrida como aceitarCorrida, que muda sua disponibilidade para false, o status da corrida para ACEITA e lança a exceção EstadoInvalidoDaCorridaException, se necessário. O método finalizarCorrida que o torna disponível novamente e o método corridaCancelada que informa também modifica a disponibilidade do motorista e informa que a corrida foi cancelada pelo usuário, caso o motorista tente cancelar a corrida que já tá cancelada, o método lança um EstadoInvalidoDaCorridaException.

### **CLASSE CNH:**

A classe cnh, é um objeto associado a Motorista, que representa a Carteira Nacional de Habilitação, recebe como atributo o número da carteira, a categoria, o anoValidade e o anoEmissão que valida se a licença está dentro do prazo de validade com o método isValida e



se a categoria da carteira permite dirigir o veículo Carro, com o método `categoriaValidaParaCarro`, se é B ou AB.

### **CLASSE VEICULO:**

A classe `Veiculo`, assim como a classe `CNH`, é um objeto associado a `Motorista` que representa o veículo a ser utilizado pelo `Motorista`. Recebe atributos como placa, cor, modelo e ano, além de um *ENUM* para a Categoria do veículo, se é `COMUM` ou `LUXO`. Possui o método `anoValido` que determina se o veículo tem um ano de fabricação aceitável para rodar, nesse caso o veículo só pode rodar se tiver no máximo 10 anos de fabricação, e claro valida se o carro não está com um ano de fabricação futuro.

### **CLASSE CORRIDA:**

A classe `Corrida` é a entidade central do sistema, ligando um `Passageiro` a um `Motorista` entre uma origem e um destino. Nela temos um *ENUM* para controlar o `StatusCorrida` como `SOLICITADA`, `ACEITA`, `EM_ANDAMENTO`, `FINALIZADA` E `CANCELADA`. Toda corrida inicia como `Solicitada`, afinal ela é criada quando meu `Passageiro` chama o método `solicitarCorrida`. Nela, o método `atribuirMotorista` verifica se há motoristas disponíveis, se tiver ela atribui esse motorista encontrado, muda o status da corrida para `ACEITA` e a disponibilidade do motorista se torna `false` e atribui essa corrida criada ao motorista. Além disso, temos o método `calcularPreco` que calcula a distância da origem pro destino, verificamos a categoria do veículo, se é `luxo` ou `comum` e retornamos o valor da corrida de acordo com a categoria. Além dos métodos de gerenciamento como `iniciar`, `cancelar` e `finalizar` `Corrida` que muda o status da corrida, disponibilidade do motorista e lança exceções se necessário.

### *Serviços*

### **CLASSE USUARIOSERVICE:**

Essa é uma classe de serviço genérica que gerencia a lista de usuários. Nela é utilizada o polimorfismo paramétrico, `Generics`, para implementar os métodos, adicionar, buscar, remover e mostrar usuários, usamos essa ferramenta para que o código seja reutilizável e funcione as mesmas ferramentas tanto para `motorista` quanto para `passageiro`, desde que herdem de usuário.



### **CLASSE PAGAMENTOSERVICE:**

Essa é uma classe abstrata que serve como base para processar qualquer tipo de pagamento. Ela define o método abstrato `processarPagamento(double valor)`, forçando que todas suas classes filhas implementem o processamento, esse é um tipo de poliformismo por sobrescrita, pois em cada subclasse é implementado de maneira única.

### **CLASSE PAGAMENTOCARTAO:**

Estende `PagamentoService` e processa pagamentos no cartão. O método `processarPagamento` herdado valida o número do cartão, tem que ter 16 dígitos e a senha de 4 dígitos, caso algo falhe ele lança o `PagamentoRecusadoException`.

### **CLASSE PAGAMENTOPIX:**

Estende `PagamentoService` e processa pagamentos no Pix. O método `processarPagamento` herdado verifica se foi informada uma chave pix e se o valor é maior que 0, de acordo com a validação, ou lança a `PagamentoRecusadoException` ou `SaldoInsuficienteException`.

### **CLASSE PAGAMENTODINHEIRO:**

Estende `PagamentoService` e processa pagamentos no dinheiro. O método `processarPagamento` herdado verifica se o valor entregue é suficiente para o valor da corrida e lança uma exceção `SaldoInsuficienteException` caso não seja suficiente.

*Utils*

### **CLASSE CALCULADORADISTANCIA:**

É uma classe utilitária que usa apenas um método estático, `calcular(String origem, String destino)` para determinar o valor da corrida. Ela utiliza o comprimento das string de origem e destino para determinar um valor numérico de representa a distância, primeiro calcula o fator que é a soma dos comprimentos das strings, multiplica por 0.5 e soma 7.0, apenas para ter um valor fictício para calcular o preço da corrida.



## 2. Conceitos Aplicados:

**Herança:** é um princípio fundamental utilizado para reutilizar códigos e estabelecer hierarquia de classes, no nosso projeto isso é aplicado nas classes motorista e passageiro que herdam atributos e métodos e usuário e na classe pagamentoCartao, pagamentoPix e pagamentoDinheiro que herdam o método da classe abstrata pagamentoService.

**Polimorfismo:** é um princípio que permite que o sistema trate objetos de forma uniforme. No nosso projeto aplicamos:

- Polimorfismo por sobrescrita: é aplicado quando redefinimos métodos nas subclasses, é aplicado em processarPagamento definido na classe base PagamentoService, que é sobrescrito nas subclasses de PagamentoCartão/Pix/Dinheiro. Evita que façamos uma classe de pagamento cheia de if/elses para cada tipo de pagamento.
- Polimorfismo paramétrico: é aplicado na classe UsuarioService <T extends Usuario>, que utiliza o Generics para criar uma classe que gerencia coleções de tipos diferentes, ou Passageiro ou Motorista para implementar nosso gerenciamento de busca e remoção, basta passar qual tipo ele quer implementar.
- Polimorfismo por sobrecarga: é aplicado na classe usuário em adicionarAvaliacao, onde o cliente pode escolher avaliar apenas com uma nota ou com uma nota e um comentário.

## 3. UML:

