1. O que o aluno fez no projeto:

Responsável pela implementação completa do sistema de locadora de veículos utilizando Programação Orientada a Objetos (POO).
Foram desenvolvidas as classes principais, a lógica de reserva, devolução, cálculo de valores, leitura dos dados, armazenamento nas listas internas e geração dos relatórios.
Também foi construído o menu interativo no terminal, permitindo que o usuário utilize todas as funcionalidades do sistema.

2. Funcionalidades implementadas pelo grupo

Todas as funcionalidades principais foram implementadas:

- Cadastro e leitura de clientes via arquivo CSV
- Cadastro e leitura de veículos via arquivo CSV
- Realização de reservas
- Devolução de veículos com cálculo automático do valor final
- Sistema de pontos de fidelidade
- Relatório de receitas da locadora
- Relatório de veículos em uso
- Consulta de pontos do cliente
- Identificação automática de veículos de luxo ou econômicos
- Armazenamento interno em listas e busca eficiente por CPF e placa

3. Funcionalidades não implementadas e justificativas

- **Tratamento avançado de erros**
  Alguns erros foram tratados, mas não houve tempo para implementar validações mais robustas (como datas inválidas, veículo já reservado etc.).
- **Interface gráfica (GUI)**
  Ficou fora do escopo por ser um projeto focado em POO e lógica.

4. Experiência do aluno

- A experiência permitiu aplicar na prática os principais conceitos de POO e entender melhor como organizar sistemas reais usando classes, objetos, herança e polimorfismo.
  A parte mais desafiadora foi a função de reservas, especialmente o processo de localizar clientes e veículos a partir dos arquivos CSV e garantir que a reserva fosse válida.

Princípios de POO Utilizados

- **Herança:** apliquei herança ao criar uma hierarquia de veículos. As classes *CarroDeLuxo* e *CarroEconomico* herdam de *Veiculo*, reutilizando atributos e métodos da classe base e evitando duplicação de código.
- **Encapsulamento:** todos os atributos das classes foram definidos como *private*, garantindo proteção dos dados e possibilitando controle de acesso por meio de métodos públicos. Isso melhora a segurança e a manutenção do código.
- **Polimorfismo:** o método *calcularValorAluguel()* é utilizado de forma polimórfica. Cada tipo de carro implementa sua própria lógica de cálculo, permitindo que o código cliente trate todos os veículos de maneira uniforme.
- **Abstração:** a classe *Veiculo* representa o modelo genérico de um veículo, definindo apenas as características essenciais. Detalhes específicos são implementados nas subclasses, simplificando o design e deixando o sistema mais modular.

O aprendizado foi significativo, especialmente no entendimento de:

- Como classes se relacionam entre si
- Como organizar um projeto maior
- Como manipular datas e dados de entrada
- Como aplicar polimorfismo de forma útil (cálculo de aluguel)

5. Bibliotecas utilizadas e motivos

O projeto utilizou as seguintes bibliotecas nativas do Java:

- **java.util.Scanner**: leitura de entrada do usuário
- **java.util.ArrayList**: armazenamento dinâmico de objetos
- **java.time.LocalDate**: manipulação de datas para reservas
- **java.io (File, BufferedReader, FileReader)** *(caso tenha usado)*: leitura dos arquivos CSV

Essas bibliotecas foram usadas por serem nativas, simples, eficientes e adequadas ao projeto.

6. Referências consultadas

- Documentação oficial da Oracle Java
- Aulas e slides fornecidos pelo professor
- Vídeos e tutoriais no YouTube sobre POO e Java
- Exemplos de leitura de CSV em Java
- Partes do código foram feitas com ajuda de IA

7. Impressão (screenshot/trecho) de uma classe importante

```
public class Locadora {  5 usages  Enzo Lacerda
    public void carregarReservas() {  1 usage  Enzo Lacerda
        try (BufferedReader br = new BufferedReader(
                new InputStreamReader(new FileInputStream(ARQUIVO_RESERVAS),  charsetName: "UTF-8"))) {

            String linha;
            while ((linha = br.readLine()) != null) {
                linha = normalizeLine(linha);
                if (linha.isEmpty()) continue;

                try {
                    String[] dados = linha.split( regex: ";");
                    if (dados.length >= 8) {
                        String id = dados[0].trim();
                        String cpf = limparCPF(dados[1].trim());
                        String placa = limparPlaca(dados[2].trim());

                        LocalDate retirada = LocalDate.parse(dados[3].trim());
                        LocalDate devolucaoPrevista = LocalDate.parse(dados[4].trim());
                        double valorEstimado = Double.parseDouble(dados[5].trim());
                        double valorPago = Double.parseDouble(dados[6].trim());
                        String status = dados[7].trim();

                        Cliente cliente = buscarClientePorCPF(cpf);
                        Veiculo veiculo = buscarCarroPorPlaca(placa);

                        if (cliente != null && veiculo != null) {
                            Reserva reserva = new Reserva(id, cliente, veiculo, retirada, devolucaoPrevista, valorEs
                            reserva.setValorPago(valorPago);
                            reserva.setStatus(status);
                            this.reservas.add(reserva);
                            contagemReservas++;
```

Uma das partes mais complicadas e desafiadoras do projeto foi o carregarReservas. Ela é responsável por ler o arquivo de reservas, validar dados, garantir consistência entre clientes e veículos e reconstruir corretamente cada reserva na memória.

Justificativa da importância

- Esse método centraliza a **recuperação do estado da aplicação**.
- Garante integridade das reservas, evitando criar objetos inconsistentes.
- Envolve **tratamento de erros**, normalização de texto, validação e lógica de negócio.

- Foi o método que mais demandou **debug**, entendimento da estrutura dos arquivos e conferência cruzada com listas de clientes e veículos.
- Representa bem o uso de **POO aplicada a um problema real**.