SISTEMA DE GERENCIAMENTO DE ALUGUEL DE VEÍCULOS

Programação e Desenvolvimento de Software II
Semestre 2/2024 - Turma TA - Grupo 6

Professor: Wagner Cipriano

Integrantes do Grupo:

Alda Letícia Rocha de Cravalho
Ana Luiza Campos Luz
Sarah Mel dos Santos Turino
Kaio Lucas Goes Costa

25 de janeiro de 2025

1 Introdução

No mundo atual, a mobilidade é um dos pilares fundamentais para a realização de atividades cotidianas, sejam elas pessoais ou profissionais. Muitas pessoas não possuem um veículo próprio ou precisam de soluções de transporte flexíveis para viagens ou compromissos específicos. A falta de opções práticas e acessíveis para alugar um veículo pode resultar em atrasos, limitações de locomoção e custos elevados com alternativas menos convenientes. Este projeto se propõe a resolver esses problemas por meio de uma locadora de carros, cujo objetivo é oferecer uma experiência eficiente, acessível e personalizada para o usuário.

Alugar um veículo traz benefícios como maior liberdade de deslocamento, flexibilidade em viagens e economia em comparação a outras opções de transporte. Por isso, a solução proposta inclui funcionalidades como: pesquisar e reservar veículos disponíveis, personalizar o período de locação, consultar informações detalhadas sobre cada veículo, gerenciar reservas ativas, e acessar um histórico de locações anteriores. Essas funcionalidades garantem aos usuários uma experiência prática e confiável ao atender suas necessidades de transporte.

2 Projeto

O programa implementado em C++, foi composto por classes modularizadas em arquivos de código-fonte e header, de forma que gere clareza e fácil manutenção. Cada componente desempenha um papel específico no funionamento do sistema.

2.1 Estrutura do Código

Veículo: Representa o veículo disponível para a locação, contendo dados como modelo, marca, câmbio, entre outros. Além de fornecer o valor da diária e exibe ao final a confirmação de cadastro no sistema.

Cliente: Representa a conta do cliente no sistema, incluindo dados como nome, email, CPF, entre outros. Permite ao usuário a criação de uma senha e confirma o cadastro no sistema.

Locação: Responsável por gerenciar os veículos disponíveis para locação.

Itens Opcionais: Representa itens adicionais que podem ser incluídos durante a locação do veículos, como rack, GPS e cadeirinha infantil.

Vistoria: Responsável por gerenciar as vistorias realizadas nos veículos, incluindo a aprovação ou reprovação e fornecendo observações sobre o status do veículo.

Reservas: Responsável por gerenciar os veículos reservados, calculado o valor total da reserva, podendo notificar o cliente sobre alguma alteração e gerando uma lista com veículos reservados.

Histórico de Transações: Responsável registrar as operações realizadas pelo cliente durante o uso do programa, contendo o histórico financeiro, a lista de pedidos e calcula o valor gasto nas locações.

3 Classes

3.1 Referência da Classe Cliente

Representa um cliente com informações pessoais.

`#include <cliente.hpp>`

Membros Públicos

- **Cliente** ()

  *Construtor padrão para a classe **Cliente*** (p. 2).
- **Cliente** (const std::string &nome, const std::string &email, const std::string &cpf, int idade, const std::string &senha)

  *Construtor parametrizado para a classe **Cliente*** (p. 2).
- void **setNome** (const std::string &nome)

  Define o nome do cliente.
- std::string **getNome** () const

  Obtém o nome do cliente.
- void **setEmail** (const std::string &email)

  Define o e-mail do cliente.
- std::string **getEmail** () const

  Obtém o e-mail do cliente.
- void **setCPF** (const std::string &cpf)

  Define o CPF do cliente.
- std::string **getCPF** () const

  Obtém o CPF do cliente.
- void **setIdade** (int idade)

  Define a idade do cliente.
- int **getIdade** () const

  Obtém a idade do cliente.
- void **setSenha** (const std::string &senha)

  Define a senha do cliente.
- std::string **getSenha** () const

  Obtém a senha do cliente.
- bool **validarDados** () const

  Valida se os dados do cliente estão completos.
- void **exibirMensagemCriacao** () const

  Exibe uma mensagem de confirmação de criação do cadastro.

3.1.1 Descrição detalhada

Representa um cliente com informações pessoais.

Esta classe armazena os dados de um cliente, como nome, e-mail, CPF, idade e senha, e fornece métodos para acessar e modificar esses dados.

3.1.2 Construtores e Destrutores

3.1.2.1 Cliente() [1/2]

```
Cliente::Cliente ()
```

Construtor padrão para a classe **Cliente** (p. 2).

Inicializa a idade como 0.

```
00010     : idade(0) {
00011 }
```

3.1.2.2 Cliente() [2/2]

```
Cliente::Cliente (
           const std::string & nome,
           const std::string & email,
           const std::string & cpf,
           int idade,
           const std::string & senha)
```

Construtor parametrizado para a classe **Cliente** (p. 2).

Parâmetros

nome	Nome do cliente.
email	E-mail do cliente.
cpf	CPF do cliente.
idade	Idade do cliente.
senha	Senha do cliente.

```
00024     : nome(nome), email(email), cpf(cpf), idade(idade), senha(senha) {
00025 }
```

3.1.3 Documentação das funções

3.1.3.1 exibirMensagemCriacao()

```
void Cliente::exibirMensagemCriacao () const
```

Exibe uma mensagem de confirmação de criação do cadastro.

Exibe uma mensagem informando que o cadastro do cliente foi criado com sucesso.

```
00098                                           {
00099     std::cout « "Cadastro do cliente '" « nome « "' criado com sucesso!" « std::endl;
00100 }
```

3.1.3.2 getCPF()

```
std::string Cliente::getCPF () const
```

Obtém o CPF do cliente.

Retorna

O CPF do cliente.

```
00068                                {
00069     return cpf;
00070 }
```

3.1.3.3 getEmail()

```
std::string Cliente::getEmail () const
```

Obtém o e-mail do cliente.

Retorna

O e-mail do cliente.

```
00059                                  {
00060     return email;
00061 }
```

3.1.3.4 getIdade()

```
int Cliente::getIdade () const
```

Obtém a idade do cliente.

Retorna

A idade do cliente.

```
00077                           {
00078     return idade;
00079 }
```

3.1.3.5 getNome()

```
std::string Cliente::getNome () const
```

Obtém o nome do cliente.

Retorna

O nome do cliente.

```
00041                                {
00042     return nome;
00043 }
```

3.1.3.6 getSenha()

```
std::string Cliente::getSenha () const
```

Obtém a senha do cliente.

Retorna

A senha do cliente.

```
00050                                 {
00051     return senha;
00052 }
```

3.1.3.7 setCPF()

```
void Cliente::setCPF (
            const std::string & cpf)
```

Define o CPF do cliente.

Parâmetros

cpf	CPF do cliente.

3.1.3.8 setEmail()

```
void Cliente::setEmail (
            const std::string & email)
```

Define o e-mail do cliente.

Parâmetros

email	E-mail do cliente.

3.1.3.9 setIdade()

```
void Cliente::setIdade (
            int idade)
```

Define a idade do cliente.

Parâmetros

idade	Idade do cliente.

3.1.3.10 setNome()

```
void Cliente::setNome (
            const std::string & nome)
```

Define o nome do cliente.

Parâmetros

nome	Nome do cliente.

```
00032                                       {
00033     this->nome = nome;
00034 }
```

3.1.3.11 setSenha()

```
void Cliente::setSenha (
            const std::string & senha)
```

Define a senha do cliente.

Parâmetros

senha	Senha do cliente.
senha	Senha a ser atribuída ao cliente.

```
00107                                               {
00108     this->senha = senha;
00109 }
```

3.1.3.12 validarDados()

```
bool Cliente::validarDados () const
```

Valida se os dados do cliente estão completos.

Verifica se os campos nome, email e CPF estão preenchidos.

Retorna

Retorna true se os dados obrigatórios estiverem preenchidos, caso contrário, false.

Verifica se os campos nome, email e CPF estão preenchidos.

Retorna

Retorna true se os dados estiverem completos, caso contrário, false.

```
00088                                   {
00089     // Verifica se os campos obrigatórios estão preenchidos
00090     return !nome.empty() && !email.empty() && !cpf.empty();
00091 }
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/ **cliente.hpp**
- SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/ **cliente.cpp**

3.2 Referência da Classe HistoricoTransacoes

Representa o histórico de transações de locações de veículos.

```
#include <historicoTransacoes.hpp>
```

Membros Públicos

- **HistoricoTransacoes** ()

  *Construtor padrão para a classe **HistoricoTransacoes** (p. 7).*
- void **registrarTransacao** (const **Cliente** &cliente, const **Locacao** &locacao)

  Registra uma transação no histórico.
- void **listarPedidos** () const

  Lista todas as transações registradas no histórico.
- void **listarPedidos** (const std::string &cpf) const

  Lista as transações de um cliente específico, filtradas pelo CPF.
- void **filtrarTransacoes** (double valorMinimo) const

  Filtra as transações por valor mínimo.
- void **armazenarDados** () const

  Armazena os dados das transações em um arquivo.
- void **recuperarDadosAnterior** ()

  Recupera dados anteriores das transações de um arquivo.

3.2.1 Descrição detalhada

Representa o histórico de transações de locações de veículos.

A classe **`HistoricoTransacoes`** (p. 7) armazena as transações realizadas pelos clientes, incluindo dados como o CPF do cliente, o valor da transação e a descrição da locação. Ela permite registrar, listar e filtrar transações.

3.2.1.1 Construtores e Destrutores

3.2.2 HistoricoTransacoes()

```
HistoricoTransacoes::HistoricoTransacoes ()
```

Construtor padrão para a classe **HistoricoTransacoes** (p. 7).

Inicializa um objeto de histórico de transações, sem transações registradas.

```
00009                                   {
00010 }
```

3.2.3 Documentação das funções

3.2.3.1 armazenarDados()

```
void HistoricoTransacoes::armazenarDados () const
```

Armazena os dados das transações em um arquivo.

Este método simula o armazenamento dos dados em um arquivo. Pode ser modificado para salvar os dados em um arquivo real.

```
00097                                                {
00098     // Se quiser salvar em arquivo...
00099     std::cout « "[Historico] Armazenando dados em arquivo...\n";
00100 }
```

3.2.3.2 filtrarTransacoes()

```
void HistoricoTransacoes::filtrarTransacoes (
          double valorMinimo) const
```

Filtra as transações por valor mínimo.

Este método exibe as transações cujo valor seja maior ou igual ao valor mínimo fornecido.

Parâmetros

valorMinimo	O valor mínimo para filtrar as transações.

```
00081                                                                    {
00082     std::cout « "[Historico] Transações com valor >= " « valorMinimo « ":\n";
00083     for (const auto& t : transacoes) {
00084         if (t.valor >= valorMinimo) {
00085             std::cout « " - CPF: " « t.clienteCPF
00086                       « ", Valor: R$ " « t.valor
00087                       « ", Desc: " « t.descricao « std::endl;
00088         }
00089     }
00090 }
```

3.2.3.3 listarPedidos() [1/2]

```
void HistoricoTransacoes::listarPedidos () const
```

Lista todas as transações registradas no histórico.

Este método exibe todas as transações para visualização de administradores ou outros usuários com permissão.

```
00039                                                {
00040     std::cout « "[Historico] Listando TODAS as transações:\n";
00041     if (transacoes.empty()) {
00042         std::cout « " - Não há transações registradas.\n";
00043         return;
00044     }
00045     for (const auto& t : transacoes) {
00046         std::cout « " - CPF: " « t.clienteCPF
00047                   « ", Valor: R$ " « t.valor
00048                   « ", Descrição: " « t.descricao « std::endl;
00049     }
00050 }
```

3.2.3.4 listarPedidos() [2/2]

```
void HistoricoTransacoes::listarPedidos (
          const std::string & cpf) const
```

Lista as transações de um cliente específico, filtradas pelo CPF.

Este método exibe as transações associadas a um determinado CPF.

Parâmetros

cpf	O CPF do cliente cujas transações devem ser listadas.

```
00059                                                                  {
00060     std::cout « "[Historico] Listando transações do CPF: " « cpf « "\n";
00061     bool encontrou = false;
00062     for (const auto& t : transacoes) {
00063         if (t.clienteCPF == cpf) {
00064             encontrou = true;
00065             std::cout « " - Valor: R$ " « t.valor
00066                       « ", Descrição: " « t.descricao « std::endl;
00067         }
00068     }
00069     if (!encontrou) {
00070         std::cout « "Nenhuma transação encontrada para este CPF.\n";
00071     }
00072 }
```

3.2.3.5 recuperarDadosAnterior()

```
void HistoricoTransacoes::recuperarDadosAnterior ()
```

Recupera dados anteriores das transações de um arquivo.

Este método simula a recuperação dos dados de transações anteriores a partir de um arquivo. Pode ser modificado para ler de um arquivo real.

```
00107                                              {
00108     // Se quiser ler de arquivo...
00109     std::cout « "[Historico] Recuperando dados anteriores...\n";
00110 }
```

3.2.3.6 registrarTransacao()

```
void HistoricoTransacoes::registrarTransacao (
          const  Cliente & cliente,
          const  Locacao & locacao)
```

Registra uma transação no histórico.

Associa uma transação com base em um cliente e uma locação de veículo, armazenando os dados da transação no histórico.

Parâmetros

cliente	O cliente que realizou a locação.
locacao	A locação de veículo realizada pelo cliente.

```
00020
      {
00021     Transacao t;
00022     t.clienteCPF = cliente.getCPF();
00023     t.valor = locacao.getValorTotal();
00024     t.descricao = "Locação do veículo: " + locacao.getVeiculoSelecionado().getModelo();
00025     transacoes.push_back(t);
00026
00027     std::cout « "[Historico] Transação registrada para o cliente: "
00028               « cliente.getNome()
00029               « " (CPF: " « t.clienteCPF « "), "
00030               « "Valor: R$ " « t.valor
00031               « ", Desc: " « t.descricao « std::endl;
00032 }
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/ **historicoTransacoes.↩ hpp**
- SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/ **historicoTransacoes.↩ cpp**

3.3 Referência da Classe ItensOpcionais

Classe responsável por gerenciar itens opcionais de locação.

`#include <itensOpcionais.hpp>`

Membros Públicos

- **ItensOpcionais** ()

  *Construtor padrão para a classe **ItensOpcionais** (*p. *11).*
- **ItensOpcionais** (bool rack, bool gps, bool cadeirinha)

  *Construtor parametrizado para a classe **ItensOpcionais** (*p. *11).*
- void **setRack** (bool rack)

  Define a seleção do rack como verdadeiro ou falso.
- bool **getRack** () const

  Obtém o estado da seleção do rack.
- void **setGPS** (bool gps)

  Define a seleção do GPS como verdadeiro ou falso.
- bool **getGPS** () const

  Obtém o estado da seleção do GPS.

- void **setCadeirinha** (bool cadeirinha)

  Define a seleção da cadeirinha como verdadeira ou falsa.
- bool **getCadeirinha** () const

  Obtém o estado da seleção da cadeirinha.
- void **adicionarItem** (const std::string &item)

  Adiciona um item opcional ao conjunto de itens selecionados.
- void **removerItem** (const std::string &item)

  Remove um item opcional do conjunto de itens selecionados.
- double **calcularValorItens** () const

  Calcula o valor total dos itens opcionais selecionados.
- void **listarItens** () const

  Exibe os itens opcionais atualmente selecionados.

3.3.1 Descrição detalhada

Classe responsável por gerenciar itens opcionais de locação.

A classe **`ItensOpcionais`** (p. 11) gerencia os itens adicionais (como rack, GPS e cadeirinha) que podem ser adicionados a uma locação. A classe permite adicionar ou remover itens, calcular o valor total dos itens selecionados e listar os itens ativos.

3.3.2 Construtores e Destrutores

3.3.2.1 ItensOpcionais() [1/2]

```
ItensOpcionais::ItensOpcionais ()
```

Construtor padrão para a classe **ItensOpcionais** (p. 11).

Inicializa os itens opcionais como não selecionados (false) e define os valores padrão dos itens: rack (R$30.0), GPS (R$50.0) e cadeirinha (R$20.0).

```
00011     : rack(false), gps(false), cadeirinha(false),
00012       valorRack(30.0), valorGps(50.0), valorCadeirinha(20.0) {
00013 }
```

3.3.2.2 ItensOpcionais() [2/2]

```
ItensOpcionais::ItensOpcionais (
            bool rack,
            bool gps,
            bool cadeirinha)
```

Construtor parametrizado para a classe **ItensOpcionais** (p. 11).

Inicializa os itens opcionais de acordo com os valores fornecidos para rack, gps e cadeirinha, e define os valores padrão dos itens: rack (R$30.0), GPS (R$50.0) e cadeirinha (R$20.0).

Parâmetros

rack	Se o rack está selecionado ou não.
gps	Se o GPS está selecionado ou não.
cadeirinha	Se a cadeirinha está selecionada ou não.

```
00026     : rack(rack), gps(gps), cadeirinha(cadeirinha),
00027       valorRack(30.0), valorGps(50.0), valorCadeirinha(20.0) {
00028 }
```

3.3.3 Documentação das funções

3.3.3.1 adicionarItem()

```
void ItensOpcionais::adicionarItem (
          const std::string & item)
```

Adiciona um item opcional ao conjunto de itens selecionados.

Dependendo do nome do item passado como argumento, o item será marcado como selecionado.

Parâmetros

item	O nome do item a ser adicionado. Pode ser "rack", "gps" ou "cadeirinha".

```
00091                                                       {
00092     if(item == "rack") setRack(true);
00093     else if(item == "gps") setGPS(true);
00094     else if(item == "cadeirinha") setCadeirinha(true);
00095 }
```

3.3.3.2 calcularValorItens()

```
double ItensOpcionais::calcularValorItens () const
```

Calcula o valor total dos itens opcionais selecionados.

Este método soma os valores dos itens que foram selecionados (marcados como true).

Retorna

O valor total dos itens selecionados.

```
00117                                                {
00118     double valor = 0.0;
00119     if(rack)       valor += valorRack;
00120     if(gps)        valor += valorGps;
00121     if(cadeirinha) valor += valorCadeirinha;
00122     return valor;
00123 }
```

3.3.3.3 getCadeirinha()

```
bool ItensOpcionais::getCadeirinha () const
```

Obtém o estado da seleção da cadeirinha.

Retorna

Retorna true se a cadeirinha está selecionada, caso contrário, false.

```
00080                                       {
00081     return cadeirinha;
00082 }
```

3.3.3.4 getGPS()

```
bool ItensOpcionais::getGPS () const
```

Obtém o estado da seleção do GPS.

Retorna

Retorna true se o GPS está selecionado, caso contrário, false.

```
00062                                   {
00063     return gps;
00064 }
```

3.3.3.5 getRack()

```
bool ItensOpcionais::getRack () const
```

Obtém o estado da seleção do rack.

Retorna

Retorna true se o rack está selecionado, caso contrário, false.

```
00044                                    {
00045     return rack;
00046 }
```

3.3.3.6 listarItens()

```
void ItensOpcionais::listarItens () const
```

Exibe os itens opcionais atualmente selecionados.

Este método exibe na tela os itens que estão marcados como selecionados (true).

```
00130                                       {
00131     std::cout « "Itens Opcionais Ativos:\n";
00132     if(rack)       std::cout « "- Rack\n";
00133     if(gps)        std::cout « "- GPS\n";
00134     if(cadeirinha) std::cout « "- Cadeirinha\n";
00135 }
```

3.3.3.7 removerItem()

```
void ItensOpcionais::removerItem (
            const std::string & item)
```

Remove um item opcional do conjunto de itens selecionados.

Dependendo do nome do item passado como argumento, o item será desmarcado como selecionado.

Parâmetros

item	O nome do item a ser removido. Pode ser "rack", "gps" ou "cadeirinha".

```
00104                                                   {
00105     if(item == "rack") setRack(false);
00106     else if(item == "gps") setGPS(false);
00107     else if(item == "cadeirinha") setCadeirinha(false);
00108 }
```

3.3.3.8 setCadeirinha()

```
void ItensOpcionais::setCadeirinha (
            bool cadeirinha)
```

Define a seleção da cadeirinha como verdadeira ou falsa.

Parâmetros

cadeirinha	Se a cadeirinha está selecionada ou não.

```
00071                                                 {
00072     this->cadeirinha = cadeirinha;
00073 }
```

3.3.3.9 setGPS()

```
void ItensOpcionais::setGPS (
            bool gps)
```

Define a seleção do GPS como verdadeiro ou falso.

Parâmetros

gps	Se o GPS está selecionado ou não.

```
00053                                   {
00054     this->gps = gps;
00055 }
```

3.3.3.10 setRack()

```
void ItensOpcionais::setRack (
           bool rack)
```

Define a seleção do rack como verdadeiro ou falso.

Parâmetros

rack	Se o rack está selecionado ou não.

```
00035                                    {
00036     this->rack = rack;
00037 }
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/ **itensOpcionais.hpp**
- SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/ **itensOpcionais.cpp**

3.4 Referência da Classe Locacao

Classe que representa uma locação de veículo, incluindo dados do cliente, veículo, itens opcionais e informações sobre o valor total da locação.

```
#include <locacao.hpp>
```

Membros Públicos

- **Locacao** ()

  *Construtor padrão para a classe **Locacao** (*p. *16).*
- **Locacao** (const std::string &dataRetirada, const std::string &dataDevolucao, const std::string &cpfCliente)

  Construtor para inicializar a locação com dados específicos de data e CPF do cliente.
- void **setDataRetirada** (const std::string &data)

  Define a data de retirada do veículo.

- std::string **getDataRetirada** () const

  Obtém a data de retirada do veículo.
- void **setDataDevolucao** (const std::string &data)

  Define a data de devolução do veículo.
- std::string **getDataDevolucao** () const

  Obtém a data de devolução do veículo.
- void **setVeiculoSelecionado** (const **Veiculo** &veiculo)

  Define o veículo selecionado para a locação.
- **Veiculo** **getVeiculoSelecionado** () const

  Obtém o veículo selecionado para a locação.
- void **setItensOpcionais** (const **ItensOpcionais** &itens)

  Define os itens opcionais selecionados para a locação.
- **ItensOpcionais** **getItensOpcionais** () const

  Obtém os itens opcionais selecionados para a locação.
- void **setValorTotal** (double valor)

  Define o valor total da locação.
- double **getValorTotal** () const

  Obtém o valor total da locação.
- void **setDevolvido** (bool dev)

  Define o status de devolução do veículo.
- bool **isDevolvido** () const

  Verifica se o veículo foi devolvido.
- void **setCpfCliente** (const std::string &cpf)

  Define o CPF do cliente responsável pela locação.
- std::string **getCpfCliente** () const

  Obtém o CPF do cliente responsável pela locação.
- void **buscarVeiculosDisponiveis** ()

  Busca os veículos disponíveis para locação.
- void **filtrarVeiculos** (const std::string &filtro)

  Filtra os veículos disponíveis com base em um critério de filtro.
- void **calcularValorReserva** ()

  Calcula o valor total da locação, considerando a diária e os itens opcionais.
- void **exibirValorReserva** () const

  Exibe o valor total da locação.
- void **confirmarReserva** ()

  Confirma a reserva do veículo.
- void **realizarVistoria** (**Vistoria** &vistoria)

  Realiza a vistoria do veículo no momento da devolução.

3.4.1 Descrição detalhada

Classe que representa uma locação de veículo, incluindo dados do cliente, veículo, itens opcionais e informações sobre o valor total da locação.

A classe gerencia todos os aspectos relacionados a uma locação de veículo, incluindo a seleção do veículo, adição de itens opcionais, cálculo do valor da locação e a devolução do veículo.

3.4.2 Construtores e Destrutores

3.4.2.1 Locacao() [1/2]

```
Locacao::Locacao ()
```

Construtor padrão para a classe **Locacao** (p. 16).

Inicializa a locação com valores padrão: valor total como 0.0 e devolução como falsa.

Inicializa o valor total como 0.0 e a condição de devolução como falsa.

```
00010     : valorTotal(0.0), devolvido(false)
00011 {
00012 }
```

3.4.2.2 Locacao() [2/2]

```
Locacao::Locacao (
            const std::string & dataRetirada,
            const std::string & dataDevolucao,
            const std::string & cpfCliente)
```

Construtor para inicializar a locação com dados específicos de data e CPF do cliente.

Construtor que recebe data e CPF do cliente.

Parâmetros

dataRetirada	Data de retirada do veículo.
dataDevolucao	Data prevista para devolução do veículo.
cpfCliente	CPF do cliente responsável pela locação.

Inicializa os dados da locação com as informações fornecidas, como data de retirada, data de devolução, CPF do cliente, e define o valor total como 0.0 e a condição de devolução como falsa.

Parâmetros

dataRetirada	Data de retirada do veículo.
dataDevolucao	Data prevista para devolução do veículo.
cpfCliente	CPF do cliente que está realizando a locação.

```
00028     : dataRetirada(dataRetirada),
00029       dataDevolucao(dataDevolucao),
00030       valorTotal(0.0),
00031       devolvido(false),
00032       cpfCliente(cpfCliente)
00033 {
00034 }
```

3.4.3 Documentação das funções

3.4.3.1 buscarVeiculosDisponiveis()

`void Locacao::buscarVeiculosDisponiveis ()`

Busca os veículos disponíveis para locação.

Exibe uma mensagem simples simulando a busca por veículos disponíveis.

Este método exibe uma mensagem simples simulando a busca por veículos disponíveis.

```
00167                                       {
00168     std::cout « "[Locacao] Buscando veículos disponíveis...\n";
00169 }
```

3.4.3.2 calcularValorReserva()

`void Locacao::calcularValorReserva ()`

Calcula o valor total da locação, considerando a diária e os itens opcionais.

Calcula o valor total da reserva.

O valor total é calculado somando o valor da diária do veículo e o valor dos itens opcionais.

Este método calcula o valor da locação somando o valor da diária do veículo selecionado e o valor dos itens opcionais.

```
00188                                   {
00189     double valorBase = veiculoSelecionado.getValorDiaria();
00190     double valorItens = itens.calcularValorItens();
00191     valorTotal = valorBase + valorItens;
00192 }
```

3.4.3.3 confirmarReserva()

```
void Locacao::confirmarReserva ()
```

Confirma a reserva do veículo.

Este método exibe uma mensagem confirmando a reserva do veículo selecionado.

```
00208                                {
00209     std::cout « "[Locacao] Reserva confirmada para o veículo: "
00210               « veiculoSelecionado.getModelo() « std::endl;
00211 }
```

3.4.3.4 exibirValorReserva()

```
void Locacao::exibirValorReserva () const
```

Exibe o valor total da locação.

Exibe o valor total da reserva.

Este método exibe o valor total calculado para a locação.

Este método exibe o valor total da locação, incluindo a diária do veículo e os itens opcionais.

```
00199                                      {
00200     std::cout « "[Locacao] Valor total da reserva: R$ " « valorTotal « std::endl;
00201 }
```

3.4.3.5 filtrarVeiculos()

```
void Locacao::filtrarVeiculos (
           const std::string & filtro)
```

Filtra os veículos disponíveis com base em um critério de filtro.

Filtra os veículos disponíveis com base no filtro fornecido.

Parâmetros

filtro	O critério utilizado para filtrar os veículos, como modelo ou tipo.

Este método exibe uma mensagem filtrando os veículos conforme o critério fornecido.

Parâmetros

filtro	O critério utilizado para filtrar os veículos, como modelo ou tipo.

```
00178                                                 {
00179     std::cout « "[Locacao] Filtrando veículos por: " « filtro « std::endl;
00180 }
```

3.4.3.6 getCpfCliente()

```
std::string Locacao::getCpfCliente () const
```

Obtém o CPF do cliente responsável pela locação.

Retorna

O CPF do cliente.

```
00158                                       {
00159     return cpfCliente;
00160 }
```

3.4.3.7 getDataDevolucao()

```
std::string Locacao::getDataDevolucao () const
```

Obtém a data de devolução do veículo.

Obtém a data de devolução da locação.

Retorna

A data de devolução do veículo.

```
00068                                         {
00069     return dataDevolucao;
00070 }
```

3.4.3.8 getDataRetirada()

```
std::string Locacao::getDataRetirada () const
```

Obtém a data de retirada do veículo.

Obtém a data de retirada da locação.

Retorna

A data de retirada do veículo.

```
00050                                       {
00051     return dataRetirada;
00052 }
```

3.4.3.9 getItensOpcionais()

```
ItensOpcionais Locacao::getItensOpcionais () const
```

Obtém os itens opcionais selecionados para a locação.

Retorna

Os itens opcionais escolhidos para a locação.

Os itens opcionais selecionados.

```
00104                                                {
00105     return itens;
00106 }
```

3.4.3.10 getValorTotal()

```
double Locacao::getValorTotal () const
```

Obtém o valor total da locação.

Retorna

O valor total da locação.

```
00122                                    {
00123     return valorTotal;
00124 }
```

3.4.3.11 getVeiculoSelecionado()

```
Veiculo Locacao::getVeiculoSelecionado () const
```

Obtém o veículo selecionado para a locação.

Retorna

O veículo selecionado.

```
00086                                             {
00087     return veiculoSelecionado;
00088 }
```

3.4.3.12 isDevolvido()

```
bool Locacao::isDevolvido () const
```

Verifica se o veículo foi devolvido.

Verifica se a locação foi devolvida.

Retorna

Retorna true se o veículo foi devolvido, caso contrário retorna false.

Retorna true se a locação foi devolvida, caso contrário, retorna false.

```
00140                                {
00141     return devolvido;
00142 }
```

3.4.3.13 realizarVistoria()

```
void Locacao::realizarVistoria (
            Vistoria & vistoria)
```

Realiza a vistoria do veículo no momento da devolução.

Este método simula uma vistoria no momento da devolução do veículo e salva os dados da vistoria.

Parâmetros

vistoria	O objeto de vistoria a ser utilizado para armazenar o resultado.

Este método simula uma vistoria, aprovando a devolução do veículo e salvando os dados da vistoria.

Parâmetros

vistoria	O objeto de vistoria a ser utilizado para armazenar o resultado.

```
00220                                                {
00221     std::cout « "[Locacao] Realizando vistoria...\n";
00222     vistoria.setResultado("Aprovado"); // ou alguma lógica
00223     vistoria.salvarDados();
00224 }
```

3.4.3.14 setCpfCliente()

```
void Locacao::setCpfCliente (
           const std::string & cpf)
```

Define o CPF do cliente responsável pela locação.

Parâmetros

cpf	O CPF do cliente.

```
00149                                                {
00150     cpfCliente = cpf;
00151 }
```

3.4.3.15 setDataDevolucao()

```
void Locacao::setDataDevolucao (
           const std::string & data)
```

Define a data de devolução do veículo.

Define a data de devolução da locação.

Parâmetros

data	Data de devolução do veículo.

```
00059                                                 {
00060     dataDevolucao = data;
00061 }
```

3.4.3.16 setDataRetirada()

```
void Locacao::setDataRetirada (
           const std::string & data)
```

Define a data de retirada do veículo.

Define a data de retirada da locação.

Parâmetros

data	Data de retirada do veículo.

```
00041                                                 {
00042     dataRetirada = data;
00043 }
```

3.4.3.17 setDevolvido()

```
void Locacao::setDevolvido (
           bool dev)
```

Define o status de devolução do veículo.

Define se a locação foi devolvida ou não.

Parâmetros

dev	Se o veículo foi devolvido (true) ou não (false).
dev	Se a locação foi devolvida (true) ou não (false).

```
00131                                   {
00132     devolvido = dev;
00133 }
```

3.4.3.18 setItensOpcionais()

```
void Locacao::setItensOpcionais (
           const  ItensOpcionais & op)
```

Define os itens opcionais selecionados para a locação.

Parâmetros

itens	Os itens opcionais escolhidos para a locação.
op	Os itens opcionais selecionados.

```
00095                                                       {
00096     itens = op;
00097 }
```

3.4.3.19 setValorTotal()

```
void Locacao::setValorTotal (
           double valor)
```

Define o valor total da locação.

Parâmetros

valor	O valor total da locação, incluindo diária e itens opcionais.
valor	O valor total da locação.

```
00113                                     {
00114     valorTotal = valor;
00115 }
```

3.4.3.20 setVeiculoSelecionado()

```
void Locacao::setVeiculoSelecionado (
           const  Veiculo & veiculo)
```

Define o veículo selecionado para a locação.

Parâmetros

veiculo	O veículo selecionado para a locação.

```
00077                                                   {
00078     veiculoSelecionado = veiculo;
00079 }
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/ **locacao.hpp**
- SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/ **locacao.cpp**

3.5 Referência da Classe Reservas

Classe responsável por gerenciar as reservas de locação de veículos.

`#include <reservas.hpp>`

Membros Públicos

- **Reservas** ()

  *Construtor padrão para a classe **Reservas** (*p. *26).*
- void **adicionarReserva** (const **Locacao** &locacao)

  Adiciona uma nova reserva à lista de reservas.
- std::vector< **Locacao** > & **getListaReservas** ()

  Retorna a referência à lista de reservas.
- void **listarCarrosReservados** (const std::string &cpf) const

  Lista as reservas de um cliente específico, identificado pelo CPF.
- void **listarCarrosReservados** () const

  Lista todas as reservas ativas no sistema (independente de CPF).
- void **alterarReserva** (int indice, const **Locacao** &novaLocacao)

  Altera uma reserva existente na lista de reservas.
- void **notificarAlteracoes** () const

  Notifica sobre as alterações feitas em uma reserva.
- double **definirValorReserva** (const **Locacao** &locacao)

  Define o valor total de uma reserva.

3.5.1 Descrição detalhada

Classe responsável por gerenciar as reservas de locação de veículos.

A classe **Reservas** (p. 26) é responsável por armazenar e manipular as reservas de veículos, incluindo a adição, listagem e modificação de reservas. Ela também permite que os dados de reservas sejam acessados e manipulados conforme a necessidade de administradores ou clientes.

3.5.2 Construtores e Destrutores

3.5.2.1 Reservas()

```
Reservas::Reservas ()
```

Construtor padrão para a classe **Reservas** (p. 26).

Inicializa a lista de reservas vazia.

```
00004                   {
00005 }
```

3.5.3 Documentação das funções

3.5.3.1 adicionarReserva()

```
void Reservas::adicionarReserva (
            const  Locacao & locacao)
```

Adiciona uma nova reserva à lista de reservas.

Esse método adiciona uma nova locação à lista interna de reservas e exibe uma mensagem indicando que a reserva foi adicionada com sucesso.

Parâmetros

locacao	A locação a ser adicionada à lista de reservas.

Esse método adiciona uma locação à lista interna de reservas e exibe uma mensagem informando que a reserva foi adicionada com sucesso.

Parâmetros

locacao	A locação que representa a reserva a ser adicionada.

```
00015                                                   {
00016     listaReservas.push_back(locacao);
00017     std::cout « "[Reservas] Reserva adicionada com sucesso.\n";
00018 }
```

3.5.3.2 alterarReserva()

```
void Reservas::alterarReserva (
           int indice,
           const  Locacao & novaLocacao)
```

Altera uma reserva existente na lista de reservas.

Esse método permite alterar uma reserva existente com base no índice da reserva na lista. O índice fornecido será usado para substituir a reserva antiga pela nova locação fornecida.

Parâmetros

indice	O índice da reserva a ser alterada.
novaLocacao	A nova locação que substituirá a reserva existente.

Esse método substitui uma reserva existente na lista, dado um índice válido e uma nova locação. Após a alteração, a função de notificação é chamada.

Parâmetros

indice	O índice da reserva a ser alterada na lista de reservas.
novaLocacao	A nova locação que substituirá a reserva existente.

```
00087                                                                   {
00088     if (indice >= 0 && indice < (int)listaReservas.size()) {
00089         listaReservas[indice] = novaLocacao;
00090         std::cout « "[Reservas] Reserva alterada com sucesso.\n";
00091         notificarAlteracoes();
00092     }
00093     else {
00094         std::cout « "[Reservas] Índice de reserva inválido.\n";
00095     }
00096 }
```

3.5.3.3 definirValorReserva()

```
double Reservas::definirValorReserva (
           const  Locacao & locacao)
```

Define o valor total de uma reserva.

Calcula o valor total de uma reserva.

Esse método retorna o valor total de uma locação com base nos detalhes da locação, como a diária do veículo e os itens opcionais adicionados.

Parâmetros

locacao	A locação para a qual o valor total será calculado.

Retorna

O valor total da reserva.

Esse método retorna o valor total de uma locação, que é calculado com base nos dados da locação, como a diária do veículo e os itens opcionais escolhidos.

Parâmetros

locacao	A locação para a qual o valor total será calculado.

Retorna

O valor total da locação.

```
00116                                                   {
00117     return locacao.getValorTotal();
00118 }
```

3.5.3.4 getListaReservas()

```
std::vector<  Locacao > & Reservas::getListaReservas ()
```

Retorna a referência à lista de reservas.

Obtém a lista de todas as reservas.

Esse método permite acesso direto à lista de reservas para manipulação externa.

Retorna

Referência ao vetor que contém todas as reservas.

Esse método retorna uma referência à lista de todas as reservas no sistema.

Retorna

A lista de reservas.

```
00027                                           {
00028     return listaReservas;
00029 }
```

3.5.3.5 listarCarrosReservados() [1/2]

```
void Reservas::listarCarrosReservados () const
```

Lista todas as reservas ativas no sistema (independente de CPF).

Esse método exibe todas as reservas que ainda estão ativas (não devolvidas), independentemente do cliente que as fez. Esse método é útil para um administrador visualizar todas as reservas.

Esse método exibe todas as reservas que ainda estão ativas (não devolvidas), independente do cliente que as fez.

```
00061                                           {
00062     std::cout « "[Reservas] Listando TODAS as reservas:\n";
00063     bool encontrou = false;
00064     for (const auto& loc : listaReservas) {
00065         if (!loc.isDevolvido()) {
00066             encontrou = true;
00067             std::cout « " - Modelo: " « loc.getVeiculoSelecionado().getModelo()
00068                       « ", CPF: " « loc.getCpfCliente()
00069                       « " (Retirada: " « loc.getDataRetirada()
00070                       « ", Devolução: " « loc.getDataDevolucao() « ")\n";
00071         }
00072     }
00073     if (!encontrou) {
00074         std::cout « "Não há veículos reservados no momento.\n";
00075     }
00076 }
```

3.5.3.6 listarCarrosReservados() [2/2]

```
void Reservas::listarCarrosReservados (
            const std::string & cpf) const
```

Lista as reservas de um cliente específico, identificado pelo CPF.

Lista os veículos reservados por um cliente específico, identificado pelo CPF.

Esse método exibe as reservas ativas (não devolvidas) de um cliente, com base no CPF fornecido. Ele é útil para que um cliente veja suas reservas atuais.

Parâmetros

cpf	O CPF do cliente cujas reservas serão listadas.

Esse método exibe as reservas ativas de um cliente, ou seja, aquelas que não foram devolvidas ainda, com base no CPF fornecido.

Parâmetros

cpf	O CPF do cliente cujas reservas serão listadas.

```
00039                                                                   {
00040     std::cout « "[Reservas] Reservas do cliente CPF = " « cpf « "\n";
00041     bool encontrou = false;
00042     for (const auto& loc : listaReservas) {
00043         if (loc.getCpfCliente() == cpf && !loc.isDevolvido()) {
00044             encontrou = true;
00045             std::cout « " - Veículo: " « loc.getVeiculoSelecionado().getModelo()
00046                       « " (Retirada: " « loc.getDataRetirada()
00047                       « ", Devolução: " « loc.getDataDevolucao() « ")\n";
00048         }
00049     }
00050     if (!encontrou) {
00051         std::cout « "Nenhuma reserva ativa para este cliente.\n";
00052     }
00053 }
```

3.5.3.7 notificarAlteracoes()

`void Reservas::notificarAlteracoes () const`

Notifica sobre as alterações feitas em uma reserva.

Notifica as alterações feitas em uma reserva.

Esse método simula o envio de uma notificação indicando que uma reserva foi alterada.

Esse método simula o envio de uma notificação informando que uma reserva foi alterada.

```
00103                                       {
00104     std::cout « "[Reservas] Notificando alterações na reserva...\n";
00105 }
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/ **reservas.hpp**
- SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/ **reservas.cpp**

3.6 Referência da Classe Veiculo

Representa um veículo disponível para locação.

`#include <veiculo.hpp>`

Membros Públicos

- **Veiculo** ()

  *Construtor padrão da classe **Veiculo** (p. 31).*
- **Veiculo** (const std::string &modelo, const std::string &marca, const std::string &categoria, const std::string &cambio, int capacidade, double valorDiaria)

  *Construtor da classe **Veiculo** (p. 31).*
- void **setModelo** (const std::string &modelo)

  Define o modelo do veículo.
- std::string **getModelo** () const

  Retorna o modelo do veículo.
- void **setMarca** (const std::string &marca)

  Define a marca do veículo.
- std::string **getMarca** () const

  Retorna a marca do veículo.
- void **setCategoria** (const std::string &categoria)

  Define a categoria do veículo.
- std::string **getCategoria** () const

  Retorna a categoria do veículo.
- void **setCambio** (const std::string &cambio)

  Define o tipo de câmbio do veículo.
- std::string **getCambio** () const

  Retorna o tipo de câmbio do veículo.
- void **setCapacidade** (int capacidade)

  Define a capacidade do veículo.
- int **getCapacidade** () const

  Retorna a capacidade do veículo.
- void **setValorDiaria** (double valor)

  Define o valor da diária do veículo.
- double **getValorDiaria** () const

  Retorna o valor da diária do veículo.
- void **setDisponivel** (bool disp)

  Define a disponibilidade do veículo para locação.
- bool **isDisponivel** () const

  Verifica se o veículo está disponível para locação.
- bool **validarCamposObrigatorios** () const

  Valida se os campos obrigatórios do veículo estão corretamente preenchidos.
- void **salvarNoSistema** () const

  Salva as informações do veículo no sistema.
- void **exibirConfirmacaoCadastro** () const

  Exibe uma mensagem de confirmação do cadastro do veículo.

3.6.1 Descrição detalhada

Representa um veículo disponível para locação.

A classe **Veiculo** (p. 31) armazena informações relacionadas ao veículo, como modelo, marca, capacidade, categoria, tipo de câmbio, valor da diária e disponibilidade. Ela também oferece métodos para validar dados, cadastrar e exibir informações do veículo.

A classe **Veiculo** (p. 31) contém informações sobre um veículo, como modelo, marca, categoria, câmbio, capacidade, valor da diária e disponibilidade. Ela também oferece métodos para manipular essas informações e validar os dados.

3.6.2 Construtores e Destrutores

3.6.2.1 Veiculo() [1/2]

```
Veiculo::Veiculo ()
```

Construtor padrão da classe **Veiculo** (p. 31).

Inicializa um objeto **Veiculo** (p. 31) com valores padrão.

```
00012 : capacidade(0), valorDiaria(0.0), disponivel(true) {}
```

3.6.3 Veiculo() [2/2]

```
Veiculo::Veiculo (
            const std::string & modelo,
            const std::string & marca,
            const std::string & categoria,
            const std::string & cambio,
            int capacidade,
            double valorDiaria)
```

Construtor da classe **Veiculo** (p. 31).

Construtor que inicializa um veículo com os dados fornecidos.

Inicializa um objeto **Veiculo** (p. 31) com os dados fornecidos.

Parâmetros

modelo	O modelo do veículo (ex: "Fusca").
marca	A marca do veículo (ex: "Volkswagen").
categoria	A categoria do veículo (ex: "Sedan", "SUV").
cambio	O tipo de câmbio do veículo (ex: "Manual", "Automático").
capacidade	A capacidade do veículo (ex: número de passageiros).
valorDiaria	O valor da diária de locação.

Este construtor permite criar um objeto **Veiculo** (p. 31) a partir dos dados fornecidos como o modelo, marca, categoria, câmbio, capacidade e valor da diária.

Parâmetros

modelo	O modelo do veículo.
marca	A marca do veículo.
categoria	A categoria do veículo (por exemplo, "SUV", "sedan", etc).
cambio	O tipo de câmbio do veículo (por exemplo, "manual", "automático").
capacidade	A capacidade do veículo (número de passageiros).
valorDiaria	O valor da diária do veículo.

```
00033     : modelo(modelo),
00034       marca(marca),
00035       categoria(categoria),
00036       cambio(cambio),
00037       capacidade(capacidade),
00038       valorDiaria(valorDiaria),
00039       disponivel(true) // por padrão verdadeiro
00040 {}
```

3.6.4 Documentação das funções

3.6.4.1 exibirConfirmacaoCadastro()

`void Veiculo::exibirConfirmacaoCadastro () const`

Exibe uma mensagem de confirmação do cadastro do veículo.

Exibe uma mensagem de confirmação de cadastro do veículo.

Este método exibe uma mensagem informando que o veículo foi cadastrado com sucesso.

Esse método exibe uma mensagem informando que o veículo foi cadastrado com sucesso.

```
00173                                                {
00174     std::cout « "Cadastro do veiculo '" « modelo « "' confirmado!" « std::endl;
00175 }
```

3.6.4.2 getCambio()

`std::string Veiculo::getCambio () const`

Retorna o tipo de câmbio do veículo.

Este método retorna o tipo de câmbio do veículo.

Retorna

O tipo de câmbio do veículo.

Esse método retorna o tipo de câmbio do veículo (por exemplo, manual ou automático).

Retorna

O tipo de câmbio do veículo.

```
00115                                    {
00116     return cambio;
00117 }
```

3.6.4.3 getCapacidade()

```
int Veiculo::getCapacidade () const
```

Retorna a capacidade do veículo.

Este método retorna a capacidade do veículo.

Retorna

A capacidade do veículo.

Esse método retorna a capacidade do veículo (número de passageiros).

Retorna

A capacidade do veículo.

```
00104                                {
00105     return capacidade;
00106 }
```

3.6.4.4 getCategoria()

```
std::string Veiculo::getCategoria () const
```

Retorna a categoria do veículo.

Este método retorna a categoria do veículo.

Retorna

A categoria do veículo.

Esse método retorna a categoria do veículo (por exemplo, "SUV", "sedan", etc).

Retorna

A categoria do veículo.

```
00126                                    {
00127     return categoria;
00128 }
```

3.6.4.5 getMarca()

```
std::string Veiculo::getMarca () const
```

Retorna a marca do veículo.

Este método retorna a marca do veículo.

Retorna

A marca do veículo.

Esse método retorna a marca do veículo.

Retorna

A marca do veículo.

```
00137                                   {
00138     return marca;
00139 }
```

3.6.4.6 getModelo()

```
std::string Veiculo::getModelo () const
```

Retorna o modelo do veículo.

Este método retorna o modelo do veículo.

Retorna

O modelo do veículo.

Esse método retorna o modelo do veículo.

Retorna

O modelo do veículo.

```
00060                                   {
00061     return modelo;
00062 }
```

3.6.4.7 getValorDiaria()

```
double Veiculo::getValorDiaria () const
```

Retorna o valor da diária do veículo.

Este método retorna o valor da diária de locação do veículo.

Retorna

O valor da diária do veículo.

Esse método retorna o valor da diária do veículo.

Retorna

O valor da diária do veículo.

```
00093                                   {
00094     return valorDiaria;
00095 }
```

3.6.4.8 isDisponivel()

```
bool Veiculo::isDisponivel () const
```

Verifica se o veículo está disponível para locação.

Este método verifica se o veículo está disponível para locação.

Retorna

Retorna true se o veículo estiver disponível, caso contrário, retorna false.

Esse método verifica se o veículo está disponível para locação.

Retorna

Retorna verdadeiro se o veículo estiver disponível, falso caso contrário.

```
00082                                   {
00083     return disponivel;
00084 }
```

3.6.4.9 salvarNoSistema()

```
void Veiculo::salvarNoSistema () const
```

Salva as informações do veículo no sistema.

Este método simula o processo de salvar os dados do veículo no sistema. No exemplo, ele apenas imprime as informações no console, mas em um sistema real isso poderia envolver salvar em um banco de dados ou arquivo.

Esse método simula o processo de salvar os dados do veículo no sistema. No exemplo, ele apenas imprime os dados no console, mas na prática, poderia salvar os dados em um banco de dados ou arquivo.

```
00163                                   {
00164     // Exemplo simples: imprime no console (poderia salvar em arquivo ou BD)
00165     std::cout « "Veiculo salvo: " « modelo « " - " « marca « std::endl;
00166 }
```

3.6.4.10 setCambio()

```
void Veiculo::setCambio (
            const std::string & cambio)
```

Define o tipo de câmbio do veículo.

Este método define o tipo de câmbio do veículo.

Parâmetros

cambio	O tipo de câmbio do veículo (ex: "Manual", "Automático").

3.6.4.11 setCapacidade()

```
void Veiculo::setCapacidade (
            int capacidade)
```

Define a capacidade do veículo.

Este método define a capacidade do veículo, ou seja, o número de pessoas que ele pode transportar.

Parâmetros

capacidade	A capacidade do veículo.

3.6.4.12 setCategoria()

```
void Veiculo::setCategoria (
          const std::string & categoria)
```

Define a categoria do veículo.

Este método define a categoria do veículo.

Parâmetros

categoria	A categoria do veículo (ex: "SUV", "Sedan").

3.6.4.13 setDisponivel()

```
void Veiculo::setDisponivel (
          bool disp)
```

Define a disponibilidade do veículo para locação.

Define a disponibilidade do veículo.

Este método define se o veículo está disponível para locação ou não.

Parâmetros

disp	Se o veículo está disponível (true) ou não (false).

Esse método define se o veículo está disponível para locação ou não.

Parâmetros

disp	O valor de disponibilidade (verdadeiro para disponível, falso para não disponível).

```
00071                                   {
00072     disponivel = disp;
00073 }
```

3.6.4.14 setMarca()

```
void Veiculo::setMarca (
          const std::string & marca)
```

Define a marca do veículo.

Este método define a marca do veículo.

Parâmetros

marca	A marca do veículo (ex: "Volkswagen").

3.6.4.15 setModelo()

```
void Veiculo::setModelo (
          const std::string & modelo)
```

Define o modelo do veículo.

Este método define o modelo do veículo.

Parâmetros

modelo	O modelo do veículo (ex: "Fusca").

Esse método define o modelo do veículo.

Parâmetros

modelo	O modelo do veículo.

```
00049                                                {
00050     this->modelo = modelo;
00051 }
```

3.6.4.16 setValorDiaria()

```
void Veiculo::setValorDiaria (
          double valor)
```

Define o valor da diária do veículo.

Este método define o valor da diária de locação do veículo.

Parâmetros

valor	O valor da diária.

3.6.4.17 validarCamposObrigatorios()

```
bool Veiculo::validarCamposObrigatorios () const
```

Valida se os campos obrigatórios do veículo estão corretamente preenchidos.

Valida os campos obrigatórios do veículo.

Este método valida se os campos principais do veículo (como modelo, marca e valor da diária) estão corretamente preenchidos.

Retorna

Retorna true se os campos obrigatórios estiverem preenchidos corretamente, caso contrário, retorna false.

Esse método valida se os campos principais do veículo (modelo, marca e valor da diária) estão corretamente preenchidos.

Retorna

Retorna verdadeiro se os campos obrigatórios estiverem preenchidos corretamente, caso contrário, retorna falso.

```
00150                                               {
00151     // Verifica se os campos principais estão preenchidos
00152     // Por exemplo, se modelo e marca não estão vazios
00153     return !modelo.empty() && !marca.empty() && valorDiaria > 0;
00154 }
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/ **veiculo.hpp**
- SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/ **veiculo.cpp**

3.7 Referência da Classe Vistoria

Representa uma vistoria realizada em um veículo.

```
#include <vistoria.hpp>
```

Membros Públicos

- **Vistoria** ()

  *Construtor padrão da classe **Vistoria** (p. 41).*
- **Vistoria** (const std::string &resultado, const std::string &data, const std::string &status)

  *Construtor da classe **Vistoria** (p. 41).*
- void **setResultado** (const std::string &resultado)

  Define o resultado da vistoria.
- std::string **getResultado** () const

  Retorna o resultado da vistoria.
- void **setData** (const std::string &data)

  Define a data da vistoria.
- std::string **getData** () const

  Retorna a data da vistoria.
- void **setStatus** (const std::string &status)

  Define o status da vistoria.
- std::string **getStatus** () const

  Retorna o status da vistoria.
- void **salvarDados** () const

  Salva os dados da vistoria.

3.7.1 Descrição detalhada

Representa uma vistoria realizada em um veículo.

Representa a vistoria de um veículo.

A classe **`Vistoria`** (p. 41) é usada para armazenar as informações de uma vistoria realizada em um veículo, incluindo o resultado, a data e o status da vistoria.

A classe **`Vistoria`** (p. 41) armazena o resultado de uma vistoria realizada em um veículo, juntamente com a data e o status da vistoria. A vistoria pode ser "Aprovada", "Reprovada" ou "Pendente", e seu status pode ser "Em Aberto", "Finalizado", etc.

3.7.2 Construtores e Destrutores

3.7.2.1 Vistoria() [1/2]

```
Vistoria::Vistoria ()
```

Construtor padrão da classe **Vistoria** (p. 41).

Inicializa a vistoria com valores padrão: resultado "Pendente", status "Em Aberto", e data vazia.

```
00012     : resultado("Pendente"), data(""), status("Em Aberto") {
00013 }
```

3.7.2.2 Vistoria() [2/2]

```
Vistoria::Vistoria (
            const std::string & resultado,
            const std::string & data,
            const std::string & status)
```

Construtor da classe **Vistoria** (p. 41).

Inicializa a vistoria com os dados fornecidos.

Parâmetros

resultado	O resultado da vistoria (ex: "Aprovado", "Reprovado").
data	A data da vistoria.
status	O status da vistoria (ex: "Em Aberto", "Finalizado").

```
00026     : resultado(resultado), data(data), status(status) {
00027 }
```

3.7.3 Documentação das funções

3.7.3.1 getData()

```
std::string Vistoria::getData () const
```

Retorna a data da vistoria.

Este método retorna a data da vistoria.

Retorna

A data da vistoria.

```
00069                                 {
00070     return data;
00071 }
```

3.7.3.2 getResultado()

```
std::string Vistoria::getResultado () const
```

Retorna o resultado da vistoria.

Este método retorna o resultado da vistoria.

Retorna

O resultado da vistoria.

```
00047                                     {
00048     return resultado;
00049 }
```

3.7.3.3 getStatus()

```
std::string Vistoria::getStatus () const
```

Retorna o status da vistoria.

Este método retorna o status da vistoria.

Retorna

O status da vistoria.

```
00091                                   {
00092     return status;
00093 }
```

3.7.3.4 salvarDados()

```
void Vistoria::salvarDados () const
```

Salva os dados da vistoria.

Salva os dados da vistoria no sistema.

Este método simula o processo de salvar os dados da vistoria, imprimindo as informações no console. Em um sistema real, isso poderia envolver salvar em um banco de dados ou arquivo.

```
00101                                   {
00102     std::cout « "Vistoria salva. Resultado: " « resultado
00103               « ", Data: " « data « ", Status: " « status « std::endl;
00104 }
```

3.7.3.5 setData()

```
void Vistoria::setData (
            const std::string & data)
```

Define a data da vistoria.

Este método define a data em que a vistoria foi realizada.

Parâmetros

data	A data da vistoria.

```
00058                                           {
00059     this->data = data;
00060 }
```

3.7.3.6 setResultado()

```
void Vistoria::setResultado (
            const std::string & resultado)
```

Define o resultado da vistoria.

Este método define o resultado da vistoria (ex: "Aprovado", "Reprovado").

Parâmetros

resultado	O resultado da vistoria.

```
00036                                                   {
00037     this->resultado = resultado;
00038 }
```

3.7.3.7 setStatus()

```
void Vistoria::setStatus (
            const std::string & status)
```

Define o status da vistoria.

Este método define o status da vistoria (ex: "Em Aberto", "Finalizado").

Parâmetros

status	O status da vistoria.

```
00080                                                {
00081     this->status = status;
00082 }
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/ **vistoria.hpp**
- SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/ **vistoria.cpp**

4 Arquivos

4.1 cliente.hpp

Ir para a documentação desse arquivo.

```
00001 #ifndef CLIENTE_HPP
00002 #define CLIENTE_HPP
00003
00004 #include <string>
00005
00012 class Cliente {
00013 private:
00014     std::string nome;
00015     std::string email;
00016     std::string cpf;
00017     int idade;
00018     std::string senha;
00019
00020 public:
```

```
00026     Cliente();
00027
00037     Cliente(const std::string& nome, const std::string& email,
00038             const std::string& cpf, int idade, const std::string& senha);
00039
00040     // Getters e Setters
00041
00047     void setNome(const std::string& nome);
00048
00054     std::string getNome() const;
00055
00061     void setEmail(const std::string& email);
00062
00068     std::string getEmail() const;
00069
00075     void setCPF(const std::string& cpf);
00076
00082     std::string getCPF() const;
00083
00089     void setIdade(int idade);
00090
00096     int getIdade() const;
00097
00103     void setSenha(const std::string& senha);
00104
00110     std::string getSenha() const;
00111
00112     // Funcionalidades
00113
00121     bool validarDados() const;
00122
00128     void exibirMensagemCriacao() const;
00129 };
00130
00131 #endif
```

4.1.1 Referência do Arquivo: *SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/cliente.cpp*

```
#include "cliente.hpp"
#include <iostream>
```

4.1.2 Referência do Arquivo: *SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/cliente.hpp*

```
#include <string>
```

Componentes

- class **Cliente**

  Representa um cliente com informações pessoais.

4.2 historicoTransacoes.hpp

Ir para a documentação desse arquivo.

```
00001 #ifndef HISTORICOTRANSACOES_HPP
00002 #define HISTORICOTRANSACOES_HPP
00003
00004 #include <string>
00005 #include <vector>
00006 #include "cliente.hpp"
00007 #include "locacao.hpp"
00008
00016 class HistoricoTransacoes {
00017 private:
00024     struct Transacao {
00025         std::string clienteCPF;
00026         double valor;
00027         std::string descricao;
00028     };
00029
00030     std::vector<Transacao> transacoes;
00031
00032 public:
00038     HistoricoTransacoes();
00039
00049     void registrarTransacao(const Cliente& cliente, const Locacao& locacao);
00050
00057     void listarPedidos() const;
00058
00066     void listarPedidos(const std::string& cpf) const;
00067
00076     void filtrarTransacoes(double valorMinimo) const;
00077
00084     void armazenarDados() const;
00085
00092     void recuperarDadosAnterior();
00093 };
00094
00095 #endif
```

4.2.1 Referência do Arquivo: *SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/historicoTransacoes.cpp*

```
#include "historicoTransacoes.hpp"
#include <iostream>
```

4.2.2 Referência do Arquivo: *SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/historicoTransacoes.hpp*

```
#include <string>
#include <vector>
#include "cliente.hpp"
#include "locacao.hpp"
```

Componentes

- class **HistoricoTransacoes**

  Representa o histórico de transações de locações de veículos.

4.3 itensOpcionais.hpp

Ir para a documentação desse arquivo.

```
00001 #ifndef ITENSOPCIONAIS_HPP
00002 #define ITENSOPCIONAIS_HPP
00003
00004 #include <string>
00005
00013 class ItensOpcionais {
00014 private:
00015     bool rack;
00016     bool gps;
00017     bool cadeirinha;
00018     double valorRack;
00019     double valorGps;
00020     double valorCadeirinha;
00021
00022 public:
00029     ItensOpcionais();
00030
00041     ItensOpcionais(bool rack, bool gps, bool cadeirinha);
00042
00048     void setRack(bool rack);
00049
00055     bool getRack() const;
00056
00062     void setGPS(bool gps);
00063
00069     bool getGPS() const;
00070
00076     void setCadeirinha(bool cadeirinha);
00077
00083     bool getCadeirinha() const;
00084
00092     void adicionarItem(const std::string& item);
00093
00101     void removerItem(const std::string& item);
00102
00110     double calcularValorItens() const;
00111
00117     void listarItens() const;
00118 };
00119
00120 #endif
```

4.3.1 Referência do Arquivo: *SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/itensOpcionais.cpp*

```
#include "itensOpcionais.hpp"
#include <iostream>
```

4.3.2 Referência do Arquivo: *SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/itensOpcionais.hpp*

```
#include <string>
```

Componentes

- class **ItensOpcionais**

  Classe responsável por gerenciar itens opcionais de locação.

4.4 locacao.hpp

Ir para a documentação desse arquivo.

```
00001 #ifndef LOCACAO_HPP
00002 #define LOCACAO_HPP
00003
00004 #include <string>
00005 #include "veiculo.hpp"
00006 #include "itensOpcionais.hpp"
00007 #include "vistoria.hpp"
00008
00018 class Locacao {
00019 private:
00020     std::string dataRetirada;
00021     std::string dataDevolucao;
00022     Veiculo veiculoSelecionado;
00023     ItensOpcionais itens;
00024     double valorTotal;
00025
00026     bool devolvido;
00027     std::string cpfCliente;
00028
00029 public:
00035     Locacao();
00036
00044     Locacao(const std::string& dataRetirada,
00045             const std::string& dataDevolucao,
00046             const std::string& cpfCliente);
00047
00053     void setDataRetirada(const std::string& data);
00054
00060     std::string getDataRetirada() const;
00061
00067     void setDataDevolucao(const std::string& data);
00068
00074     std::string getDataDevolucao() const;
00075
00081     void setVeiculoSelecionado(const Veiculo& veiculo);
00082
00088     Veiculo getVeiculoSelecionado() const;
00089
00095     void setItensOpcionais(const ItensOpcionais& itens);
00096
00102     ItensOpcionais getItensOpcionais() const;
00103
00109     void setValorTotal(double valor);
00110
00116     double getValorTotal() const;
00117
```

```
00123     void setDevolvido(bool dev);
00124
00130     bool isDevolvido() const;
00131
00137     void setCpfCliente(const std::string& cpf);
00138
00144     std::string getCpfCliente() const;
00145
00151     void buscarVeiculosDisponiveis();
00152
00158     void filtrarVeiculos(const std::string& filtro);
00159
00165     void calcularValorReserva();
00166
00172     void exibirValorReserva() const;
00173
00179     void confirmarReserva();
00180
00189     void realizarVistoria(Vistoria& vistoria);
00190 };
00191
00192 #endif
```

4.4.1 Referência do Arquivo: *SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/locacao.cpp*

```
#include "locacao.hpp"
#include <iostream>
```

4.4.2 Referência do Arquivo: *SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/locacao.hpp*

```
#include <string>
#include "veiculo.hpp"
#include "itensOpcionais.hpp"
#include "vistoria.hpp"
```

Componentes

- class **Locacao**

  Classe que representa uma locação de veículo, incluindo dados do cliente, veículo, itens opcionais e informações sobre o valor total da locação.

4.5 reservas.hpp

Ir para a documentação desse arquivo.

```
00001 #ifndef RESERVAS_HPP
00002 #define RESERVAS_HPP
00003
00004 #include <vector>
00005 #include "locacao.hpp"
00006
00016 class Reservas {
00017 private:
00018     std::vector<Locacao> listaReservas;
00019
00020 public:
00026     Reservas();
00027
00036     void adicionarReserva(const Locacao& locacao);
00037
00045     std::vector<Locacao>& getListaReservas();
00046
00055     void listarCarrosReservados(const std::string& cpf) const;
00056
00063     void listarCarrosReservados() const;
00064
00074     void alterarReserva(int indice, const Locacao& novaLocacao);
00075
00081     void notificarAlteracoes() const;
00082
00092     double definirValorReserva(const Locacao& locacao);
00093 };
00094
00095 #endif
```

4.5.1 Referência do Arquivo: *SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/reservas.cpp*

```
#include "reservas.hpp"
#include <iostream>
```

4.5.2 Referência do Arquivo: *SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/reservas.hpp*

```
#include <vector>
#include "locacao.hpp"
```

Componentes

- class **Reservas**

  Classe responsável por gerenciar as reservas de locação de veículos.

4.6 veiculo.hpp

Ir para a documentação desse arquivo.

```
00001 #ifndef VEICULO_HPP
00002 #define VEICULO_HPP
00003
00004 #include <string>
00005
00014 class Veiculo {
00015 private:
00016     std::string modelo;
00017     std::string marca;
00018     std::string categoria;
00019     std::string cambio;
00020     int capacidade;
00021     double valorDiaria;
00022     bool disponivel;
00023
00024 public:
00030     Veiculo();
00031
00044     Veiculo(const std::string& modelo,
00045             const std::string& marca,
00046             const std::string& categoria,
00047             const std::string& cambio,
00048             int capacidade,
00049             double valorDiaria);
00050
00058     void setModelo(const std::string& modelo);
00059
00067     std::string getModelo() const;
00068
00076     void setMarca(const std::string& marca);
00077
00085     std::string getMarca() const;
00086
00094     void setCategoria(const std::string& categoria);
00095
00103     std::string getCategoria() const;
00104
00112     void setCambio(const std::string& cambio);
00113
00121     std::string getCambio() const;
00122
00131     void setCapacidade(int capacidade);
00132
00140     int getCapacidade() const;
00141
00149     void setValorDiaria(double valor);
00150
00158     double getValorDiaria() const;
00159
00167     void setDisponivel(bool disp);
00168
00176     bool isDisponivel() const;
00177
00187     bool validarCamposObrigatorios() const;
00188
00196     void salvarNoSistema() const;
00197
00203     void exibirConfirmacaoCadastro() const;
00204 };
00205
00206 #endif
```

4.6.1 Referência do Arquivo: *SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/veiculo.cpp*

```
#include "veiculo.hpp"
#include <iostream>
```

4.6.2 Referência do Arquivo: *SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/veiculo.hpp*

```
#include <string>
```

Componentes

- class **Veiculo**

  Representa um veículo disponível para locação.

4.7 vistoria.hpp

```
00001 #ifndef VISTORIA_HPP
00002 #define VISTORIA_HPP
00003
00004 #include <string>
00005
00014 class Vistoria {
00015 private:
00016     std::string resultado;
00017     std::string data;
00018     std::string status;
00019
00020 public:
00027     Vistoria();
00028
00038     Vistoria(const std::string& resultado, const std::string& data,
00039              const std::string& status);
00040
00048     void setResultado(const std::string& resultado);
00049
00057     std::string getResultado() const;
00058
00066     void setData(const std::string& data);
00067
00075     std::string getData() const;
00076
00084     void setStatus(const std::string& status);
00085
00093     std::string getStatus() const;
00094
00101     void salvarDados() const;
00102 };
00103
00104 #endif
```

4.7.1 Referência do Arquivo: *SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/vistoria.cpp*

```
#include "vistoria.hpp"
#include <iostream>
```

4.7.2 Referência do Arquivo: *SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/vistoria.hpp*

```
#include <string>
```

Componentes

- class **Vistoria**

  Representa uma vistoria realizada em um veículo.

4.8 main.cpp

Funções

- void **limparBuffer** ()
- int **buscarClientePorCPF** (const std::string &cpf)
- int **buscarVeiculoDisponivelPorModelo** (const std::string &modelo)
- void **listarVeiculos** ()
- void **menuAdministrador** ()
- void **devolverVeiculo** (int indiceCliente)
- void **menuClienteAutenticado** (int indiceCliente)
- void **menuCliente** ()
- int **main** ()

Variáveis

- std::vector< **Cliente** > **listaClientes**
- std::vector< **Veiculo** > **listaVeiculos**
- **Reservas** **gerenciadorReservas**
- **HistoricoTransacoes** **historicoTransacoes**

4.8.1 buscarClientePorCPF()

```
int buscarClientePorCPF (
          const std::string & cpf)
00039                                                {
00040     for (size_t i = 0; i < listaClientes.size(); i++) {
00041         if (listaClientes[i].getCPF() == cpf) {
00042             return static_cast<int>(i);
00043         }
00044     }
00045     return -1;
00046 }
```

4.8.2 buscarVeiculoDisponivelPorModelo()

```
int buscarVeiculoDisponivelPorModelo (
          const std::string & modelo)
00049                                                                {
00050     for (size_t i = 0; i < listaVeiculos.size(); i++) {
00051         // Se o modelo bate e o veículo está disponível
00052         if (listaVeiculos[i].getModelo() == modelo && listaVeiculos[i].isDisponivel()) {
00053             return static_cast<int>(i);
00054         }
00055     }
00056     return -1;
00057 }
```

4.8.3 devolverVeiculo()

```
void devolverVeiculo (
          int indiceCliente)
00230                                          {
00231     // Pede ao usuário qual veículo quer devolver
00232     std::cout « "Informe o modelo do veículo que deseja devolver: ";
00233     limparBuffer();
00234     std::string modelo;
00235     std::getline(std::cin, modelo);
00236
00237     // Precisamos encontrar a reserva correspondente (se existir)
00238     // e marcar como devolvida, realizar vistoria e liberar o veículo
00239     std::vector<Locacao>& reservas = gerenciadorReservas.getListaReservas();
00240     bool encontrada = false;
00241
00242     for (auto &loc : reservas) {
00243         if (loc.getCpfCliente() == listaClientes[indiceCliente].getCPF() &&
00244             loc.getVeiculoSelecionado().getModelo() == modelo &&
00245             !loc.isDevolvido())
00246         {
00247             // Achamos a locação do cliente com esse carro
00248             encontrada = true;
00249
00250             // Marca como devolvida
00251             loc.setDevolvido(true);
00252
00253             // Deixa o veículo disponível novamente
00254             for (auto &v : listaVeiculos) {
00255                 if (v.getModelo() == modelo) {
00256                     v.setDisponivel(true);
00257                     break;
```

```
00258                 }
00259             }
00260
00261             // Realiza vistoria
00262             Vistoria vist;
00263             vist.setData("2025-01-25"); // Exemplo de data ou use data atual
00264             loc.realizarVistoria(vist);
00265
00266             std::cout « "Veículo devolvido com sucesso e vistoria realizada.\n";
00267             break;
00268         }
00269     }
00270
00271     if (!encontrada) {
00272         std::cout « "Nenhuma reserva encontrada para esse veículo.\n";
00273     }
00274 }
```

4.8.4 limparBuffer()

```
void limparBuffer ()
00035                     {
00036     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00037 }
```

4.8.5 listarVeiculos()

```
void listarVeiculos ()
00060                       {
00061     std::cout « "\n===== Veículos Disponíveis =====" « std::endl;
00062     bool encontrou = false;
00063
00064     for (size_t i = 0; i < listaVeiculos.size(); i++) {
00065         if (listaVeiculos[i].isDisponivel()) {
00066             encontrou = true;
00067             std::cout « i « ") " « listaVeiculos[i].getModelo() « " - "
00068                       « listaVeiculos[i].getMarca() « " | Categoria: "
00069                       « listaVeiculos[i].getCategoria() « " | Câmbio: "
00070                       « listaVeiculos[i].getCambio() « " | Cap: "
00071                       « listaVeiculos[i].getCapacidade() « " pessoas | Diária: R$"
00072                       « listaVeiculos[i].getValorDiaria() « "\n";
00073         }
00074     }
00075
00076     if (!encontrou) {
00077         std::cout « "Nenhum veículo disponível no momento.\n";
00078     }
00079     std::cout « std::endl;
00080 }
```

4.8.6 main()

```
int main ()
00514           {
00515     // Já coloca alguns veículos iniciais, se quiser
00516     listaVeiculos.push_back(
00517         Veiculo("Civic", "Honda", "Sedan", "Automático", 5, 180.0));
00518     listaVeiculos.push_back(
```

```
00519         Veiculo("Corolla", "Toyota", "Sedan", "Automático", 5, 200.0));
00520     listaVeiculos.push_back(
00521         Veiculo("Gol", "Volkswagen", "Hatch", "Manual", 5, 90.0));
00522     listaVeiculos.push_back(
00523         Veiculo("Compass", "Jeep", "SUV", "Automático", 5, 220.0));
00524     listaVeiculos.push_back(
00525         Veiculo("Onix", "Chevrolet", "Hatch", "Automático", 5, 120.0));
00526     listaVeiculos.push_back(
00527         Veiculo("Strada", "Fiat", "Picape", "Manual", 2, 110.0));
00528     listaVeiculos.push_back(
00529         Veiculo("Foguete", "Nasa", "Espacial", "Automático", 1, 9999.0));
00530
00531     int opcaoGeral = -1;
00532     while(opcaoGeral != 0) {
00533         std::cout « "\n========== SISTEMA DE LOCADORA ==========\n";
00534         std::cout « "1. Cliente\n";
00535         std::cout « "2. Administrador\n";
00536         std::cout « "0. Sair\n";
00537         std::cout « "=========================================\n";
00538         std::cout « "Opção: ";
00539         std::cin » opcaoGeral;
00540
00541         switch(opcaoGeral) {
00542             case 1:
00543                 menuCliente();
00544                 break;
00545             case 2:
00546                 menuAdministrador();
00547                 break;
00548             case 0:
00549                 std::cout « "Encerrando o programa...\n";
00550                 break;
00551             default:
00552                 std::cout « "Opção inválida.\n";
00553                 break;
00554         }
00555     }
00556
00557     return 0;
00558 }
```

4.8.7 menuAdministrador()

```
void menuAdministrador ()
00086                         {
00087     std::string user, pass;
00088     std::cout « "\n==== Login de Administrador ====\n";
00089     std::cout « "Usuário: ";
00090     std::cin » user;
00091     std::cout « "Senha: ";
00092     std::cin » pass;
00093
00094     if (user != "admin" || pass != "123") {
00095         std::cout « "Usuário ou senha inválidos.\n";
00096         return;
00097     }
00098
00099     int opcaoAdmin = -1;
00100
00101     while (opcaoAdmin != 0) {
00102         std::cout « "\n===== Menu do Administrador =====\n";
00103         std::cout « "1. Cadastrar novo veículo\n";
00104         std::cout « "2. Listar veículos disponíveis\n";
00105         std::cout « "3. Cadastrar novo cliente\n";
00106         std::cout « "4. Listar todos os clientes\n";
00107         std::cout « "5. Listar histórico de transações (todos os clientes)\n"; // NOVO
00108         std::cout « "6. Listar todos os veículos reservados\n";                // NOVO
00109         std::cout « "0. Voltar ao menu principal\n";
```

```
00110         std::cout « "================================\n";
00111         std::cout « "Opção: ";
00112         std::cin » opcaoAdmin;
00113 
00114         switch(opcaoAdmin) {
00115             case 1: {
00116                 // Cadastro de novo veículo
00117                 std::string modelo, marca, categoria, cambio;
00118                 int capacidade;
00119                 double valorDiaria;
00120 
00121                 std::cout « "\n=== Cadastro de Veículo ===\n";
00122                 limparBuffer();
00123                 std::cout « "Modelo: ";
00124                 std::getline(std::cin, modelo);
00125 
00126                 std::cout « "Marca: ";
00127                 std::getline(std::cin, marca);
00128 
00129                 std::cout « "Categoria: ";
00130                 std::getline(std::cin, categoria);
00131 
00132                 std::cout « "Câmbio (Manual/Automático): ";
00133                 std::getline(std::cin, cambio);
00134 
00135                 std::cout « "Capacidade: ";
00136                 std::cin » capacidade;
00137 
00138                 std::cout « "Valor da diária: ";
00139                 std::cin » valorDiaria;
00140 
00141                 Veiculo v(modelo, marca, categoria, cambio, capacidade, valorDiaria);
00142                 if (!v.validarCamposObrigatorios()) {
00143                     std::cout « "Erro: campos obrigatórios inválidos!\n";
00144                 } else {
00145                     listaVeiculos.push_back(v);
00146                     v.salvarNoSistema();
00147                     v.exibirConfirmacaoCadastro();
00148                 }
00149                 break;
00150             }
00151             case 2: {
00152                 listarVeiculos();
00153                 break;
00154             }
00155             case 3: {
00156                 // Cadastro de novo cliente
00157                 std::string nome, email, cpf, senha;
00158                 int idade;
00159 
00160                 std::cout « "\n=== Cadastro de Cliente ===\n";
00161                 limparBuffer();
00162                 std::cout « "Nome: ";
00163                 std::getline(std::cin, nome);
00164                 std::cout « "CPF: ";
00165                 std::cin » cpf;
00166                 std::cout « "Idade: ";
00167                 std::cin » idade;
00168                 std::cout « "E-mail: ";
00169                 std::cin » email;
00170                 std::cout « "Senha: ";
00171                 std::cin » senha;
00172 
00173                 // Verifica se já existe esse CPF
00174                 if (buscarClientePorCPF(cpf) != -1) {
00175                     std::cout « "Já existe cliente com este CPF.\n";
00176                     break;
00177                 }
00178 
00179                 Cliente c(nome, email, cpf, idade, senha);
00180                 if (c.validarDados()) {
00181                     listaClientes.push_back(c);
00182                     c.exibirMensagemCriacao();
```

```
00183                 } else {
00184                     std::cout « "Dados inválidos. Não foi possível cadastrar o
      cliente.\n";
00185                 }
00186                 break;
00187             }
00188             case 4: {
00189                 std::cout « "\n===== Lista de Clientes =====\n";
00190                 if (listaClientes.empty()) {
00191                     std::cout « "Nenhum cliente cadastrado.\n";
00192                 } else {
00193                     for (size_t i = 0; i < listaClientes.size(); i++) {
00194                         std::cout « i « ") Nome: " « listaClientes[i].getNome()
00195                                   « " | CPF: " « listaClientes[i].getCPF()
00196                                   « " | Email: " « listaClientes[i].getEmail() « "\n";
00197                     }
00198                 }
00199                 std::cout « std::endl;
00200                 break;
00201             }
00202             case 5: {
00203                 // NOVO: Listar histórico de todos
00204                 std::cout « "\n=== Histórico de Transações (TODOS os clientes) ===\n";
00205                 historicoTransacoes.listarPedidos(); // Sem CPF => lista tudo
00206                 break;
00207             }
00208             case 6: {
00209                 // NOVO: Listar todos os veículos reservados (todas as reservas ativas)
00210                 std::cout « "\n=== Veículos Reservados (TODOS) ===\n";
00211                 gerenciadorReservas.listarCarrosReservados(); // Sem parâmetro => lista
      tudo
00212                 break;
00213             }
00214             case 0:
00215                 std::cout « "Retornando ao menu principal...\n";
00216                 break;
00217             default:
00218                 std::cout « "Opção inválida.\n";
00219                 break;
00220         }
00221     }
00222 }
```

4.8.8 menuCliente()

```
void menuCliente ()
```

```
00436                     {
00437     int opcaoCliente = -1;
00438     while(opcaoCliente != 0) {
00439         std::cout « "\n===== Menu de Acesso (Cliente) =====\n";
00440         std::cout « "1. Criar conta\n";
00441         std::cout « "2. Autenticar\n";
00442         std::cout « "0. Voltar ao menu principal\n";
00443         std::cout « "==================================\n";
00444         std::cout « "Opção: ";
00445         std::cin » opcaoCliente;
00446 
00447         switch(opcaoCliente) {
00448             case 1: {
00449                 std::string nome, email, cpf, senha;
00450                 int idade;
00451                 std::cout « "\n=== Criar Conta de Cliente ===\n";
00452                 limparBuffer();
00453                 std::cout « "Nome: ";
00454                 std::getline(std::cin, nome);
00455                 std::cout « "CPF: ";
00456                 std::cin » cpf;
00457                 std::cout « "Idade: ";
```

```
00458                 std::cin » idade;
00459                 std::cout « "E-mail: ";
00460                 std::cin » email;
00461                 std::cout « "Senha: ";
00462                 std::cin » senha;
00463 
00464                 if (buscarClientePorCPF(cpf) != -1) {
00465                     std::cout « "Já existe um cliente com este CPF.\n";
00466                     break;
00467                 }
00468 
00469                 Cliente novoCliente(nome, email, cpf, idade, senha);
00470                 if (novoCliente.validarDados()) {
00471                     listaClientes.push_back(novoCliente);
00472                     novoCliente.exibirMensagemCriacao();
00473                 } else {
00474                     std::cout « "Dados inválidos. Não foi possível criar conta.\n";
00475                 }
00476                 break;
00477             }
00478             case 2: {
00479                 std::string cpf, senha;
00480                 std::cout « "\n=== Autenticação de Cliente ===\n";
00481                 std::cout « "CPF: ";
00482                 std::cin » cpf;
00483                 std::cout « "Senha: ";
00484                 std::cin » senha;
00485 
00486                 int indice = buscarClientePorCPF(cpf);
00487                 if (indice == -1) {
00488                     std::cout « "Cliente não encontrado.\n";
00489                     break;
00490                 }
00491 
00492                 if (listaClientes[indice].getSenha() == senha) {
00493                     std::cout « "Autenticação realizada com sucesso!\n";
00494                     menuClienteAutenticado(indice);
00495                 } else {
00496                     std::cout « "Senha incorreta.\n";
00497                 }
00498                 break;
00499             }
00500             case 0:
00501                 std::cout « "Retornando ao menu principal...\n";
00502                 break;
00503             default:
00504                 std::cout « "Opção inválida.\n";
00505                 break;
00506         }
00507     }
00508 }
```

4.8.9 menuClienteAutenticado()

```
void menuClienteAutenticado (
          int indiceCliente)
00276                                                 {
00277     int opcao = -1;
00278     while (opcao != 0) {
00279         std::cout « "\n===== Menu do Cliente =====\n";
00280         std::cout « "1. Alterar senha\n";
00281         std::cout « "2. Listar veículos disponíveis\n";
00282         std::cout « "3. Calcular valor de uma locação\n";
00283         std::cout « "4. Reservar um veículo\n";
00284         std::cout « "5. Exibir minhas reservas atuais\n";  // (CORREÇÃO 1)
00285         std::cout « "6. Exibir meu histórico de transações\n"; // (CORREÇÃO 1)
00286         std::cout « "7. Devolver um veículo\n"; // (CORREÇÃO 5)
00287         std::cout « "0. Voltar ao menu anterior\n";
```

```
00288         std::cout « "==========================\n";
00289         std::cout « "Opção: ";
00290         std::cin » opcao;
00291 
00292         switch(opcao) {
00293             case 1: {
00294                 std::string novaSenha;
00295                 std::cout « "Digite a nova senha: ";
00296                 std::cin » novaSenha;
00297                 listaClientes[indiceCliente].setSenha(novaSenha);
00298                 std::cout « "Senha alterada com sucesso!\n";
00299                 break;
00300             }
00301             case 2: {
00302                 // Listar veículos disponíveis
00303                 listarVeiculos();
00304                 break;
00305             }
00306             case 3: {
00307                 // Calcular valor de uma locação (exemplo simples)
00308                 std::string modelo;
00309                 int dias;
00310                 limparBuffer();
00311                 std::cout « "Modelo do veículo: ";
00312                 std::getline(std::cin, modelo);
00313                 std::cout « "Quantidade de dias: ";
00314                 std::cin » dias;
00315 
00316                 int idxV = buscarVeiculoDisponivelPorModelo(modelo);
00317                 if (idxV == -1) {
00318                     std::cout « "Veículo não encontrado ou indisponível.\n";
00319                 } else {
00320                     Locacao loc;
00321                     loc.setVeiculoSelecionado(listaVeiculos[idxV]);
00322                     loc.calcularValorReserva(); // base para 1 dia
00323                     double valorUmDia = loc.getValorTotal();
00324 
00325                     double valorTotal = valorUmDia * dias;
00326                     std::cout « "Valor estimado para " « dias
00327                               « " dias: R$ " « valorTotal « "\n";
00328                 }
00329                 break;
00330             }
00331             case 4: {
00332                 // (CORREÇÃO 3 e 4) Reservar um veículo com itens opcionais e datas
    reais
00333                 std::string modelo;
00334                 limparBuffer();
00335                 std::cout « "Modelo do veículo: ";
00336                 std::getline(std::cin, modelo);
00337 
00338                 // Vamos pedir data real de retirada e devolução
00339                 std::string dataRetirada, dataDevolucao;
00340                 std::cout « "Data de Retirada (formato YYYY-MM-DD): ";
00341                 std::cin » dataRetirada;
00342                 std::cout « "Data de Devolucao (formato YYYY-MM-DD): ";
00343                 std::cin » dataDevolucao;
00344 
00345                 // Dias não é mais tão necessário (poderíamos calcular com base nas
    datas),
00346                 // mas mantemos se quiser um cálculo simples
00347                 int dias;
00348                 std::cout « "Quantidade de dias (estimado): ";
00349                 std::cin » dias;
00350 
00351                 int idxV = buscarVeiculoDisponivelPorModelo(modelo);
00352                 if (idxV == -1) {
00353                     std::cout « "Veículo não encontrado ou indisponível.\n";
00354                 } else {
00355                     // Criar uma locação
00356                     Locacao novaLocacao(dataRetirada, dataDevolucao,
00357                                         listaClientes[indiceCliente].getCPF());
00358 
```

```
00359                     novaLocacao.setVeiculoSelecionado(listaVeiculos[idxV]);
00360
00361                     // ========== Itens opcionais ==========
00362                     ItensOpcionais itens;
00363                     char resp;
00364                     std::cout « "Deseja adicionar rack? (S/N): ";
00365                     std::cin » resp;
00366                     if (resp == 'S' || resp == 's') itens.adicionarItem("rack");
00367
00368                     std::cout « "Deseja adicionar GPS? (S/N): ";
00369                     std::cin » resp;
00370                     if (resp == 'S' || resp == 's') itens.adicionarItem("gps");
00371
00372                     std::cout « "Deseja adicionar cadeirinha? (S/N): ";
00373                     std::cin » resp;
00374                     if (resp == 'S' || resp == 's') itens.adicionarItem("cadeirinha");
00375
00376                     novaLocacao.setItensOpcionais(itens);
00377
00378                     // Calcular valor (1 dia base + itens opcionais)
00379                     // e multiplicar pelos dias estimados
00380                     novaLocacao.calcularValorReserva();
00381                     double valorUmDia = novaLocacao.getValorTotal();
00382                     double valorTotal = valorUmDia * dias;
00383
00384                     std::cout « "Valor total da reserva para " « dias
00385                               « " dias: R$ " « valorTotal « "\n";
00386                     std::cout « "Confirmar reserva? (S/N): ";
00387                     char c;
00388                     std::cin » c;
00389                     if (c == 'S' || c == 's') {
00390                         // Confirma locação
00391                         novaLocacao.confirmarReserva();
00392
00393                         // Marca veículo como indisponível (CORREÇÃO 2)
00394                         listaVeiculos[idxV].setDisponivel(false);
00395
00396                         // Adicionar ao gerenciador de reservas
00397                         gerenciadorReservas.adicionarReserva(novaLocacao);
00398
00399                         // Registrar no histórico de transações
00400
    historicoTransacoes.registrarTransacao(listaClientes[indiceCliente],
00401                                                                 novaLocacao);
00402
00403                         std::cout « "Reserva concluída com sucesso!\n";
00404                     }
00405                 }
00406                 break;
00407             }
00408             case 5: {
00409                 // (CORREÇÃO 1) Exibir apenas reservas do cliente logado
00410                 gerenciadorReservas.listarCarrosReservados(
00411                     listaClientes[indiceCliente].getCPF()
00412                 );
00413                 break;
00414             }
00415             case 6: {
00416                 // (CORREÇÃO 1) Listar histórico só deste cliente
00417                 std::cout « "\n=== Seu Histórico de Transações ===\n";
00418
    historicoTransacoes.listarPedidos(listaClientes[indiceCliente].getCPF());
00419                 break;
00420             }
00421             case 7: {
00422                 // (CORREÇÃO 5) Devolver um veículo
00423                 devolverVeiculo(indiceCliente);
00424                 break;
00425             }
00426             case 0:
00427                 std::cout « "Retornando ao menu anterior...\n";
00428                 break;
00429             default:
```

```
00430                 std::cout « "Opção inválida.\n";
00431                 break;
00432         }
00433     }
00434 }
```

4.8.10 Referência do Arquivo: *SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/main.cpp*

```
#include <iostream>
#include <locale>
#include <string>
#include <vector>
#include <limits>
#include "veiculo.hpp"
#include "cliente.hpp"
#include "locacao.hpp"
#include "itensOpcionais.hpp"
#include "vistoria.hpp"
#include "reservas.hpp"
#include "historicoTransacoes.hpp"
```

4.9 test_main

Funções

- **TEST** (VeiculoTest, TestCadastroVeiculo)
- **TEST** (ClienteTest, TestCadastroCliente)
- **TEST** (ReservaTest, TestReservarVeiculo)
- **TEST** (HistoricoTransacoesTest, TestRegistrarTransacao)
- int **main** (int argc, char ∗∗argv)

4.9.1 main()

```
int main (
            int argc,
            char ** argv)
00082                                {
00083     ::testing::InitGoogleTest(&argc, argv);
00084
00085     // Rodar os testes unitários
00086     return RUN_ALL_TESTS();
00087 }
```

4.9.2 TEST() [1/4]

```
TEST (
        ClienteTest ,
        TestCadastroCliente )
00024                                   {
00025     Cliente cliente("João", "joao@email.com", "123456789", 30, "senha123");
00026
00027     // Verificar se o cliente foi cadastrado corretamente
00028     EXPECT_EQ(cliente.getNome(), "João");
00029     EXPECT_EQ(cliente.getEmail(), "joao@email.com");
00030     EXPECT_EQ(cliente.getCPF(), "123456789");
00031     EXPECT_EQ(cliente.getIdade(), 30);
00032     EXPECT_EQ(cliente.getSenha(), "senha123");
00033 }
```

4.9.3 TEST() [2/4]

```
TEST (
        HistoricoTransacoesTest ,
        TestRegistrarTransacao )
00061                                                   {
00062     Cliente cliente("Ana", "ana@email.com", "123123123", 28, "senha123");
00063     Veiculo veiculo("Fusca", "Volkswagen", "Sedan", "Manual", 5, 100.0);
00064
00065     // Adicionar cliente e veículo
00066     listaClientes.push_back(cliente);
00067     listaVeiculos.push_back(veiculo);
00068
00069     // Criar uma locação e registrar no histórico
00070     Locacao loc;
00071     loc.setVeiculoSelecionado(veiculo);
00072     loc.setCpfCliente(cliente.getCPF());
00073     loc.calcularValorReserva();
00074
00075     historicoTransacoes.registrarTransacao(cliente, loc);
00076
00077     // Testar se a transação foi registrada
00078     EXPECT_EQ(historicoTransacoes.getTransacoes().size(), 1);
00079     EXPECT_EQ(historicoTransacoes.getTransacoes()[0].getCliente().getCPF(),
    "123123123");
00080 }
```

4.9.4 TEST() [3/4]

```
TEST (
        ReservaTest ,
        TestReservarVeiculo )
00036                                   {
00037     // Criar um cliente e um veículo
00038     Cliente cliente("Maria", "maria@email.com", "987654321", 25, "senha123");
00039     Veiculo veiculo("Gol", "Volkswagen", "Hatch", "Manual", 5, 90.0);
00040
00041     // Adicionar o cliente e o veículo a suas listas
00042     listaClientes.push_back(cliente);
00043     listaVeiculos.push_back(veiculo);
00044
```

```
00045     // Reservar um veículo
00046     Locacao loc;
00047     loc.setVeiculoSelecionado(veiculo);
00048     loc.setCpfCliente(cliente.getCPF());
00049     loc.calcularValorReserva();
00050
00051     // Confirmar reserva
00052     gerenciadorReservas.adicionarReserva(loc);
00053
00054     // Testar se a reserva foi realizada
00055     EXPECT_EQ(gerenciadorReservas.getListaReservas().size(), 1);
00056     EXPECT_EQ(gerenciadorReservas.getListaReservas()[0].getCpfCliente(), "987654321");
00057
      EXPECT_EQ(gerenciadorReservas.getListaReservas()[0].getVeiculoSelecionado().getModelo(),
      "Gol");
00058 }
```

4.9.5 TEST() [4/4]

```
TEST (
          VeiculoTest ,
          TestCadastroVeiculo )
00011                                    {
00012     Veiculo veiculo("Civic", "Honda", "Sedan", "Automático", 5, 180.0);
00013
00014     // Verificar se o veículo foi cadastrado corretamente
00015     EXPECT_EQ(veiculo.getModelo(), "Civic");
00016     EXPECT_EQ(veiculo.getMarca(), "Honda");
00017     EXPECT_EQ(veiculo.getCategoria(), "Sedan");
00018     EXPECT_EQ(veiculo.getCambio(), "Automático");
00019     EXPECT_EQ(veiculo.getCapacidade(), 5);
00020     EXPECT_EQ(veiculo.getValorDiaria(), 180.0);
00021 }
```

4.9.6 Referência do Arquivo: *SISTEMA DE GERENCIAMENTO DE ALGUEL DE VEÍCULOS/test_main.cpp*

```
#include <gtest/gtest.h>
#include "veiculo.hpp"
#include "cliente.hpp"
#include "locacao.hpp"
#include "itensOpcionais.hpp"
#include "vistoria.hpp"
#include "reservas.hpp"
#include "historicoTransacoes.hpp"
```

5 Exemplo de Código

Aqui está um exemplo de como funciona o cadastro de um novo cliente.

```
#include <iostream>
#include <string>

class Cliente {
public:
    std::string nome;
    std::string cpf;
    std::string email;
    int idade;

    void cadastrarCliente() {
        std::cout << "Digite o nome: ";
        std::getline(std::cin, nome);
        std::cout << "Digite o CPF: ";
        std::cin >> cpf;
        std::cout << "Digite o e-mail: ";
        std::cin >> email;
        std::cout << "Digite a idade: ";
        std::cin >> idade;
    }
};
```

Este código é responsável por criar um novo cliente no sistema. O método `cadastrarCliente` coleta os dados do cliente e os armazena nas variáveis correspondentes.

6 Conclusão

Este projeto teve como objetivo o desenvolvimento de um sistema de locação de veículos robusto e eficiente, integrando diversas funcionalidades essenciais, como o cadastro de clientes, veículos, reservas e o gerenciamento do histórico de transações. Ele oferece uma solução simples e eficaz para o gerenciamento de locação de veículos, proporcionando uma interface amigável para os administradores e clientes, com funcionalidades completas para cadastro, reserva e devolução de veículos.

Durante o processo de desenvolvimento, foi possível aplicar de maneira prática conceitos avançados de programação orientada a objetos, incluindo encapsulamento, abstração, polimorfismo e modularidade. Como resultado, obteve-se uma solução estruturada e funcional. O principal aprendizado deste projeto foi a importância de adotar boas práticas de programação, especialmente no que diz respeito à integração entre classes e à implementação de métodos. Além disso, a resolução de desafios, como a compatibilização de diferentes módulos e a validação de entradas, contribuiu para o aprimoramento das nossas habilidades em lidar com problemas comuns no desenvolvimento de software real.

Por fim, como perspectiva para versões futuras, o sistema pode ser expandido para incluir uma interface gráfica intuitiva, proporcionando uma experiência de uso mais acessível e amigável ao usuário.

Referências

- *Documentação Doxygen*, disponível em: `https://www.doxygen.nl`.
- *Programa completo em C++*, disponível em: `https://en.cppreference.com`.
- *Repositório - PDS2-20242-TA-6-SISTEMA-DE-GERENCIAMENTO-DE-ALGUEL-DE-VE-CULOS*, disponível em: GitHub
- *Pasta HTML do Doxygen - Google Drive*, disponível em: GERENCIAMENTO DE ALGUEL DE VEÍCULOS