

***UFRJ – UNIVERSIDADE FEDERAL DO RIO DE JANEIRO***

**CCMN – Centro de Ciências Matemáticas e da Natureza**

***DCC – Departamento de Ciência da Computação***

***MAB533 – 2828 - Fundamentos de Engenharia de Software***

**Sistema Você-Aluga** |

Rio de Janeiro – RJ

Dez-2014

## **Sistema Você-Aluga**

**Equipe / Grupo:** Carlos Eduardo da Silva Martins

Diego Machado de Souza

Fernando Siqueira Rocha Gouvêa

Lucas Chaves Balabram

**Professores:** Eber Assis Schmitz & Sildenir Alves Ribeiro

## **Sumário**

5 - Introdução
6 - Arquitetura e Modelagem
12 - Sobre o Trabalho Desenvolvido
40 - Sobre os Testes aplicados
58 - Considerações Finais

## **1. Introdução**

O sistema Você-Aluga é uma empresa do grupo Você, que possui também uma rede de hotéis e uma empresa aérea, e que compartilham uma mesma base de clientes. O sistema será uma aplicação Desktop, sem componente web, que será usada pelos agentes de venda e gerentes para executar suas atividades diárias na empresa.

No sistema Você-Aluga, os agentes de venda poderão cadastrar novos clientes, fazer reservas ou alocações diretas de veículos e registrar os pagamentos dos clientes. Um gerente poderá, além de executar as atividades dos agentes de vendas, cadastrar e agendar manutenção para novos automóveis, além de cadastrar novos agentes de vendas.

### **1.1 Equipe**

Para ajudar a equipe com o desenvolvimento do sistema, utilizamos uma ferramenta que nos serviu para melhorar a visualização das tarefas e funcionalidades a serem feitas, as que estamos fazendo no momento e as que já fizemos. Essa ferramenta se chama Trello e é muito popular entre equipes e empresas que aplicam técnicas e metodologias ágeis.

Para a criação dos diagramas UML foi usado o ArgoUML.

## 2. Arquitetura e Modelagem

### 2.1 Arquitetura

#### 2.1.1 - MVC

O sistema será baseado na arquitetura MVC (*Model-View-Controller*), que determina a separação das camadas de modelo (banco de dados e classes dos objetos manipulados pelo sistema), vista (telas) e controladores (partes que executam a lógica da aplicação).

- **Modelo:** O modelo consiste no banco de dados do sistema, juntamente com as classes que representam os objetos do banco na aplicação (classes de modelo), como por exemplo as classes para Pessoa e Carro. Também fazem parte do modelo os objetos de acesso de dados (DAO - Data Access Object), que concentram a interação com o banco de dados, de modo a isolar as consultas do resto da aplicação.
- **Vista:** A vista consiste nas telas da aplicação, que será a interface com o usuário. No sistema Você-Aluga serão as telas da aplicação desktop, sem componentes web.
- **Controladores:** Os controladores consistem nos componentes que executam as regras de negócio do sistema, fazendo a interação entre as camadas de vista e modelo. No sistema Você-Aluga, essa camada será composta pelos validadores, que garantiram que as informações recebidas da vista estão de acordo com as regras de negócio antes de serem passadas à camada de modelo, e os

controladores, que executarão a lógica da aplicação interagindo com as outras camadas.

### **2.1.2 - Perfis**

Tanto os clientes quanto atendentes e gerentes serão cadastrados como pessoa física no banco do mesmo modo, e será usado um sistema de perfis para diferenciá-los.

Existirão objetos de Perfil (ex: Cliente, Atendente, Gerente) que serão usados pelo sistema para garantir os devidos privilégios para os usuários. Para cada usuário existirá um objeto “Permissão” que liga uma pessoa física a um perfil, de modo que o sistema possa identificar que operações podem ser executadas por cada pessoa.

## **2.2 - Tecnologias usadas**

### **2.2.1 - Sistema**

A interface gráfica será feita usando Swing, o toolkit padrão do Java para esse tipo de aplicação, e o plugin Window Builder do Eclipse.

O banco de dados usado será o MariaDB (MySQL), usando biblioteca Hibernate por meio do JPA (Java Persistence API) para fazer a persistência dos objetos do Java no banco de dados, de forma a facilitar o desenvolvimento e tornar o banco de dados mais fiel às classes de modelo do sistema.

Os testes serão feitos usando a framework TestNG, com mocks feitos usando a framework Mockito. Estas escolhas foram feitas baseadas na experiência prévia da equipe de desenvolvimento com essas ferramentas. Outro fator que pesou na escolha do TestNG foi que ele faz testes unitários e funcionais, enquanto o JUnit precisa de um plugin adicional para fazer testes que precisem interagir com o banco de dados.

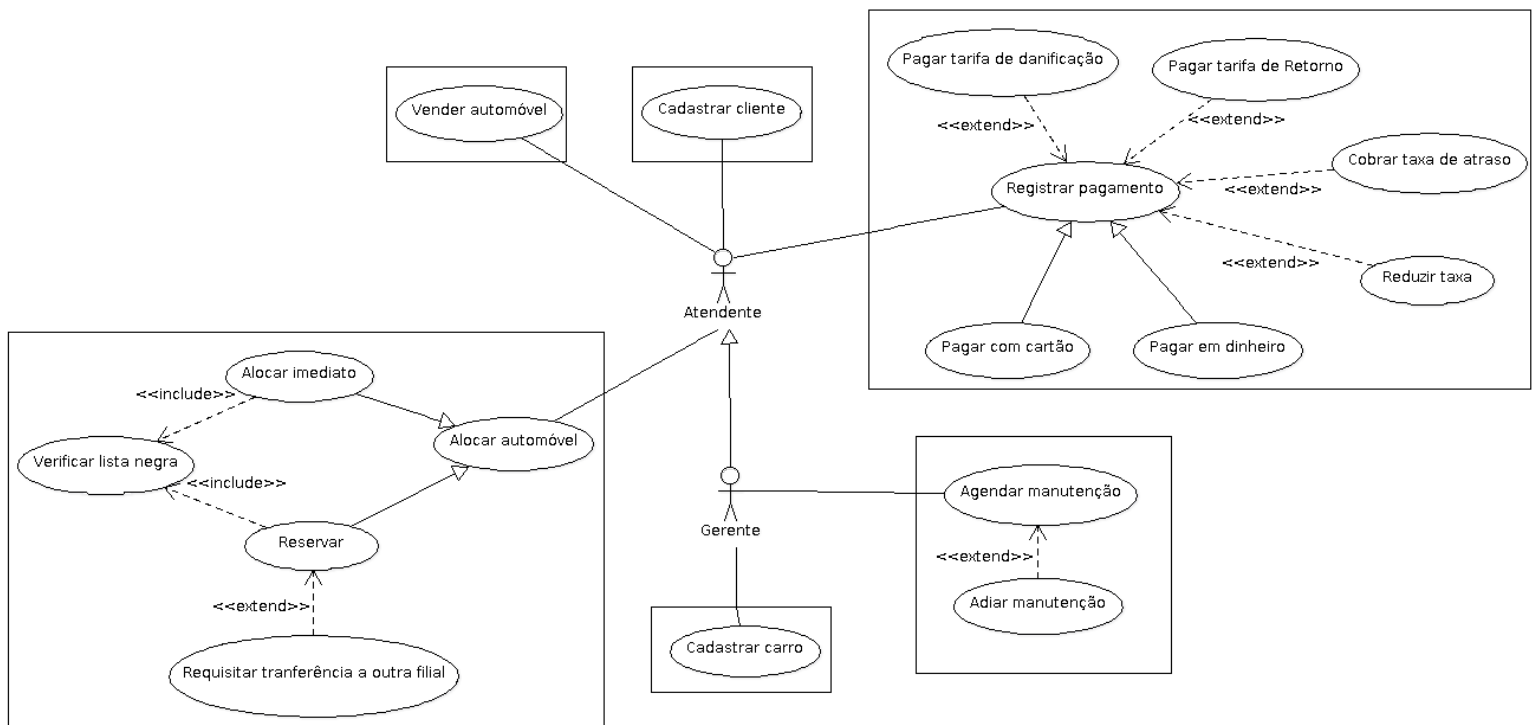
Para o versionamento do sistema usamos o Git com o apoio do Github para termos uma visualização online e user friendly do projeto.

Para a implementação do sistema utilizamos uma prática ágil para desenvolvimento de sistemas chamada TDD(Test Driven Development). Nela a

primeira coisa que é feita é a implementação dos casos de teste mesmo não tendo nenhum código implementado.

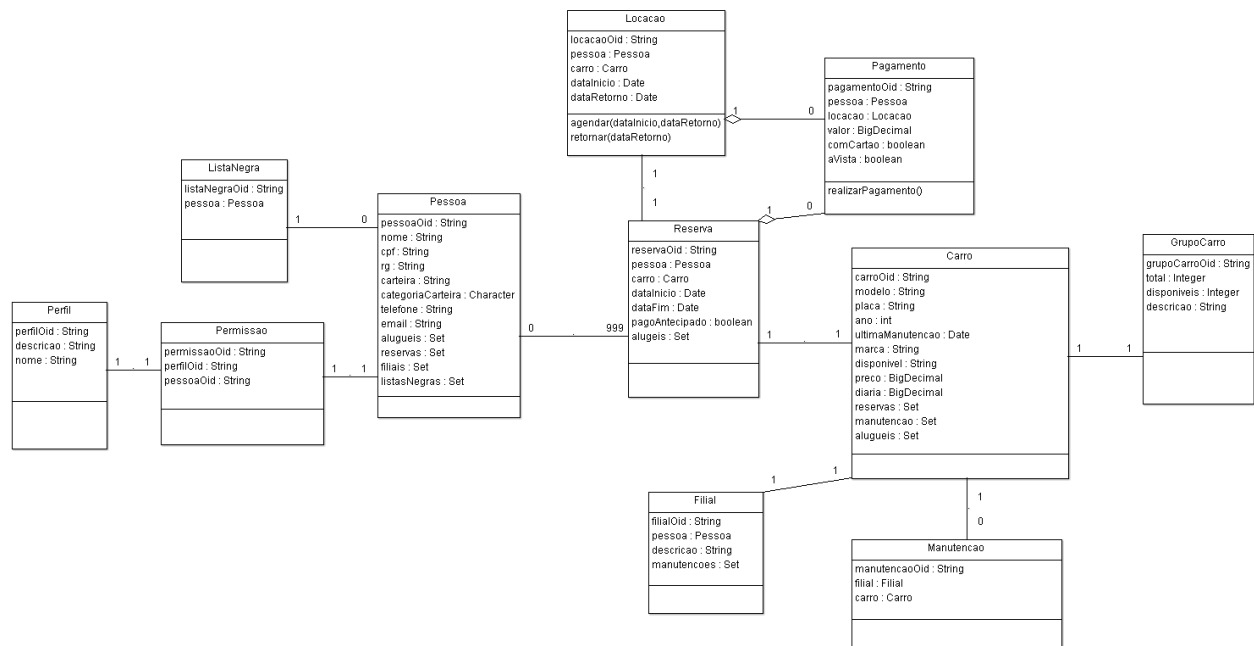
## 2.3 Diagramas

### 2.3.1 Diagramas de caso de uso

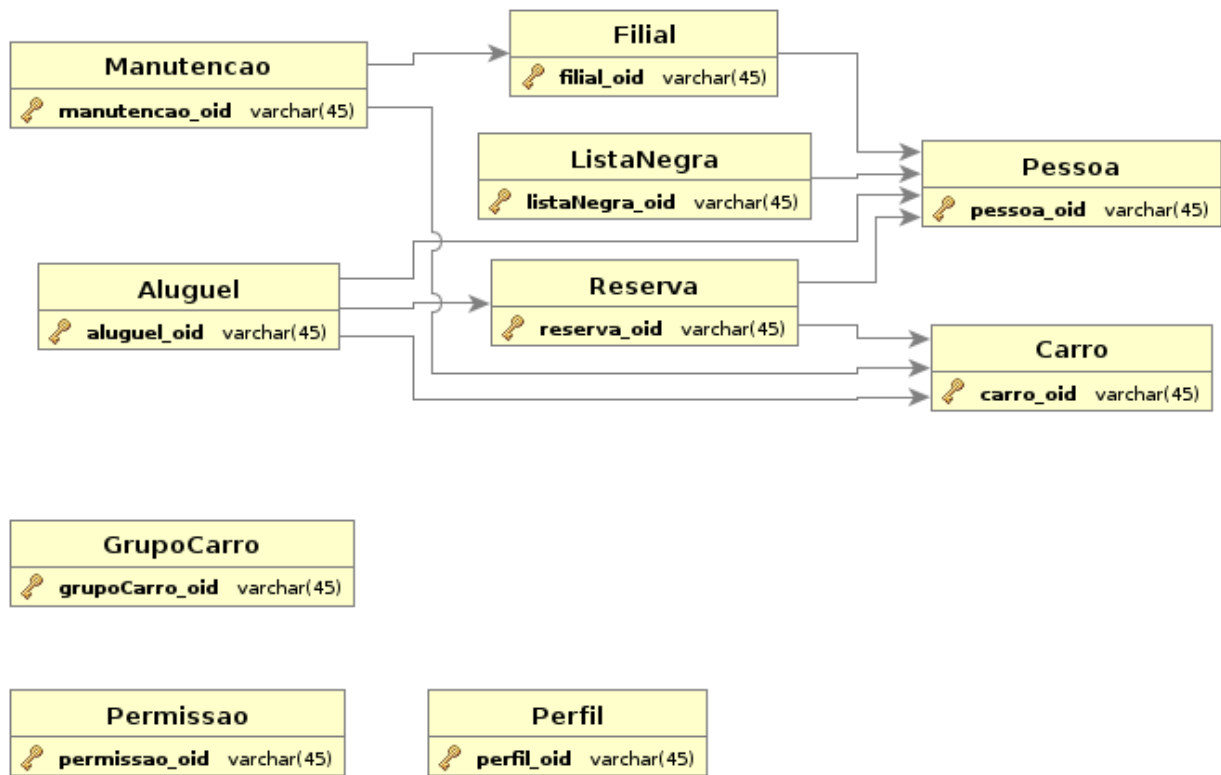




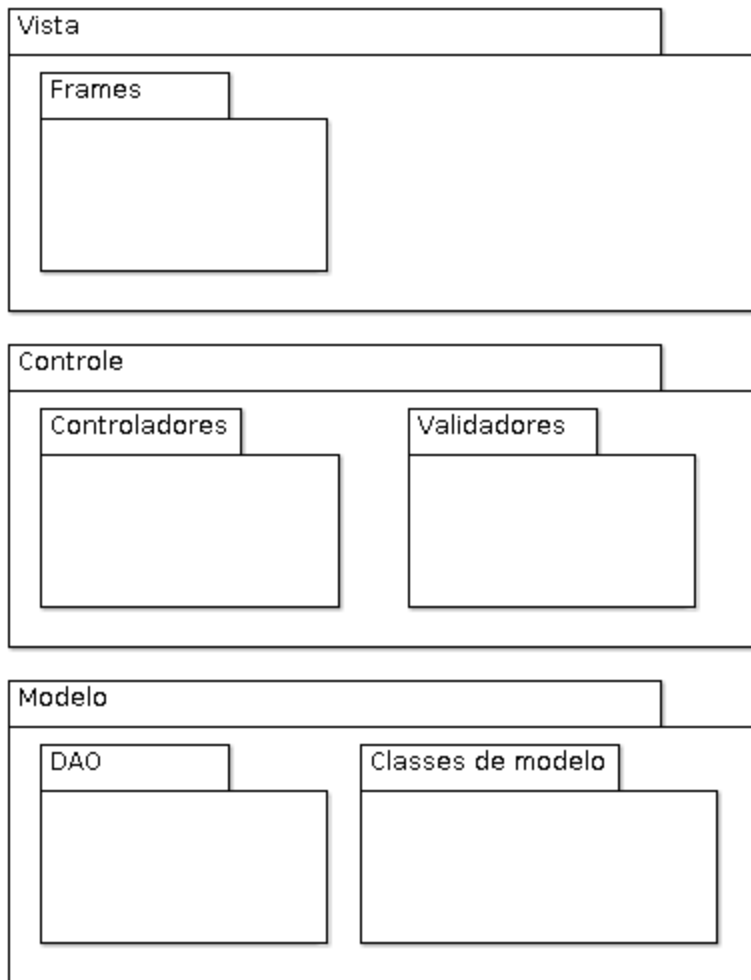
## 2.3.2 Diagrama de Classes



### 2.3.3 Banco de Dados



### 2.3.4 Diagrama MVC



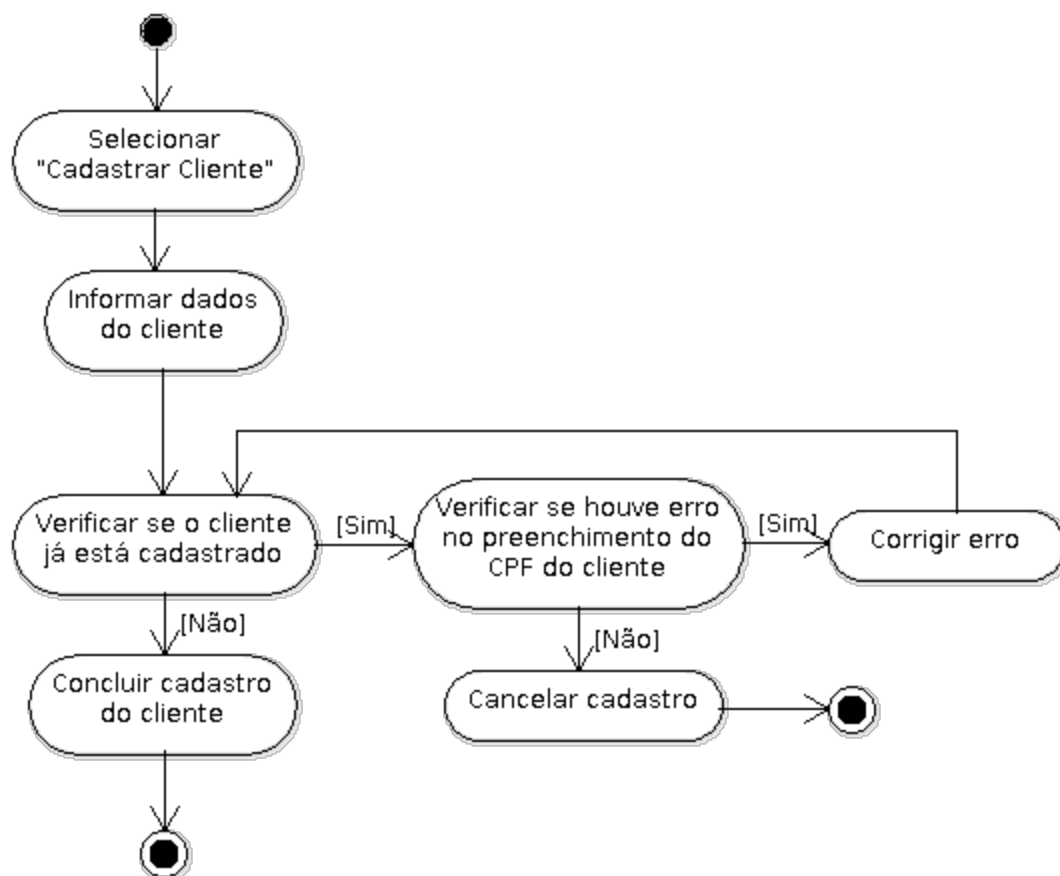
### 3. Sobre o Trabalho Desenvolvido

#### 3.1. Funcionalidades

Nesta seção serão detalhadas as funcionalidades do sistema Você-Aluga.

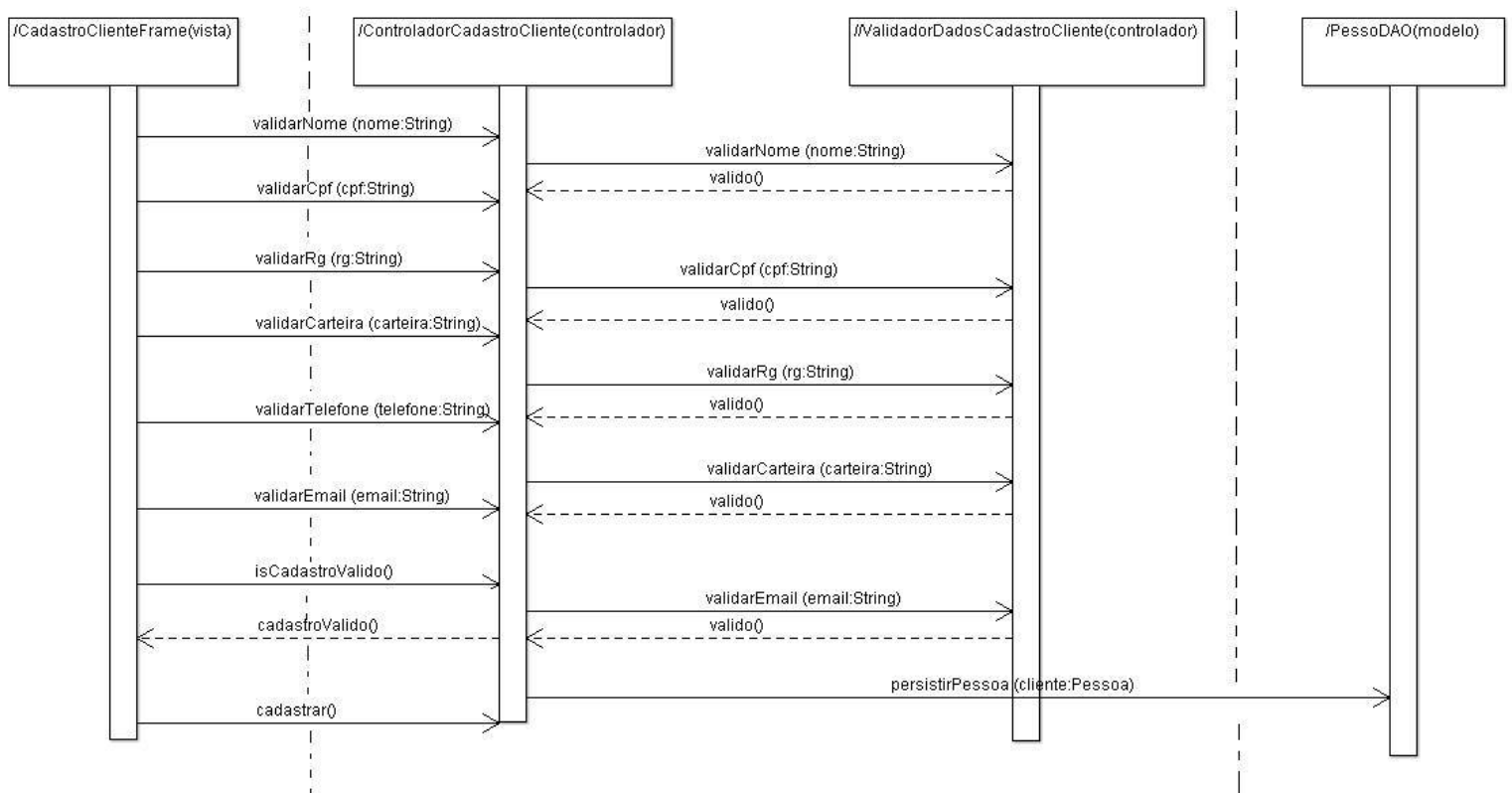
##### 3.1.1 - Cadastro de Cliente

Um atendente ou gerente pode cadastrar um novo cliente no sistema. Cada cliente só pode ser cadastrado uma vez, por isso antes de concluir o cadastro, deve ser checado se o cliente já está cadastrado e caso ele já esteja, verificar se houve erro no preenchimento do CPF, corrigindo se necessário.



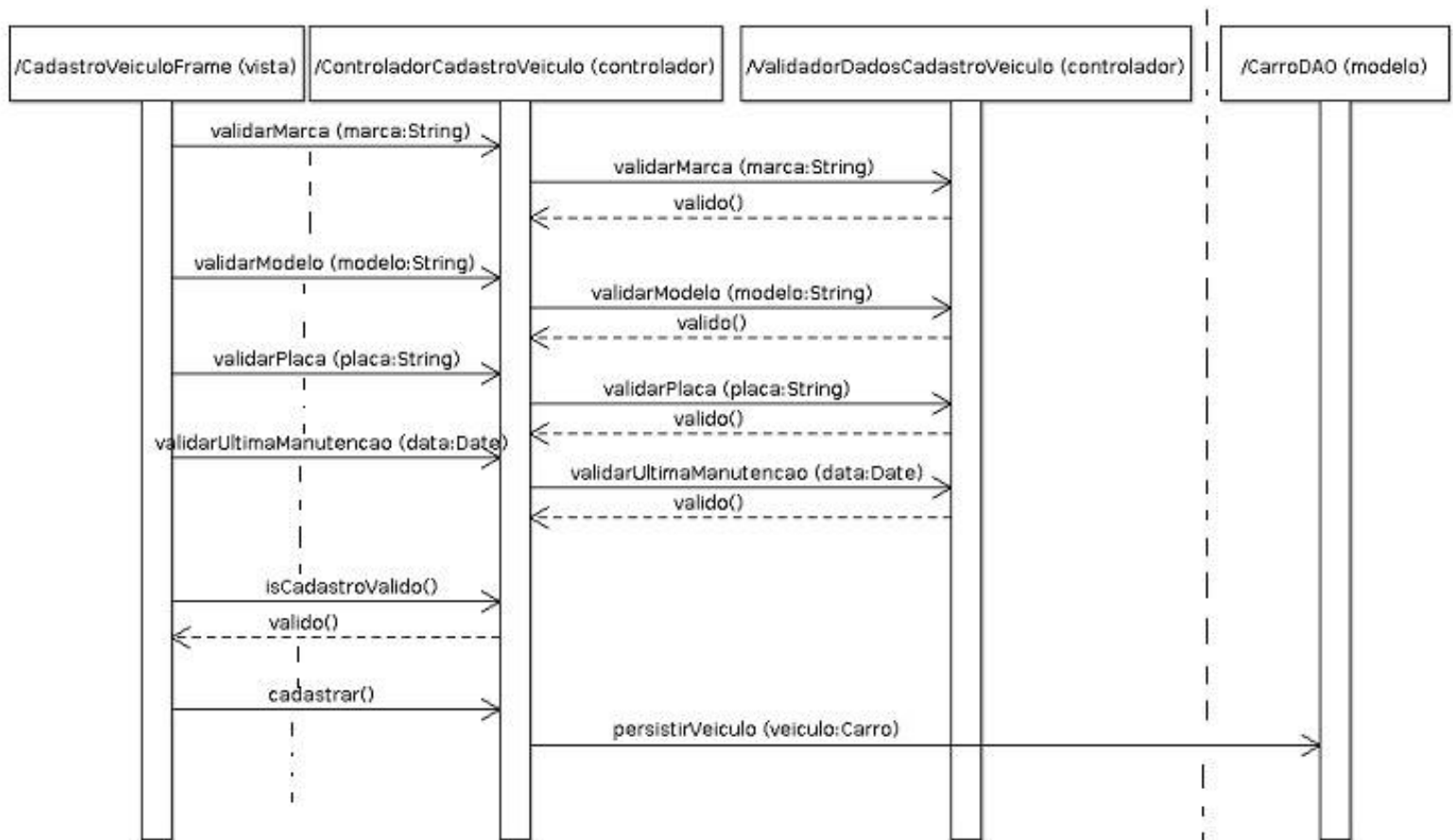
## Cadastrar cliente

Diagrama de sequência para o cadastro de clientes



### 3.1.2 - Cadastro de Automóvel

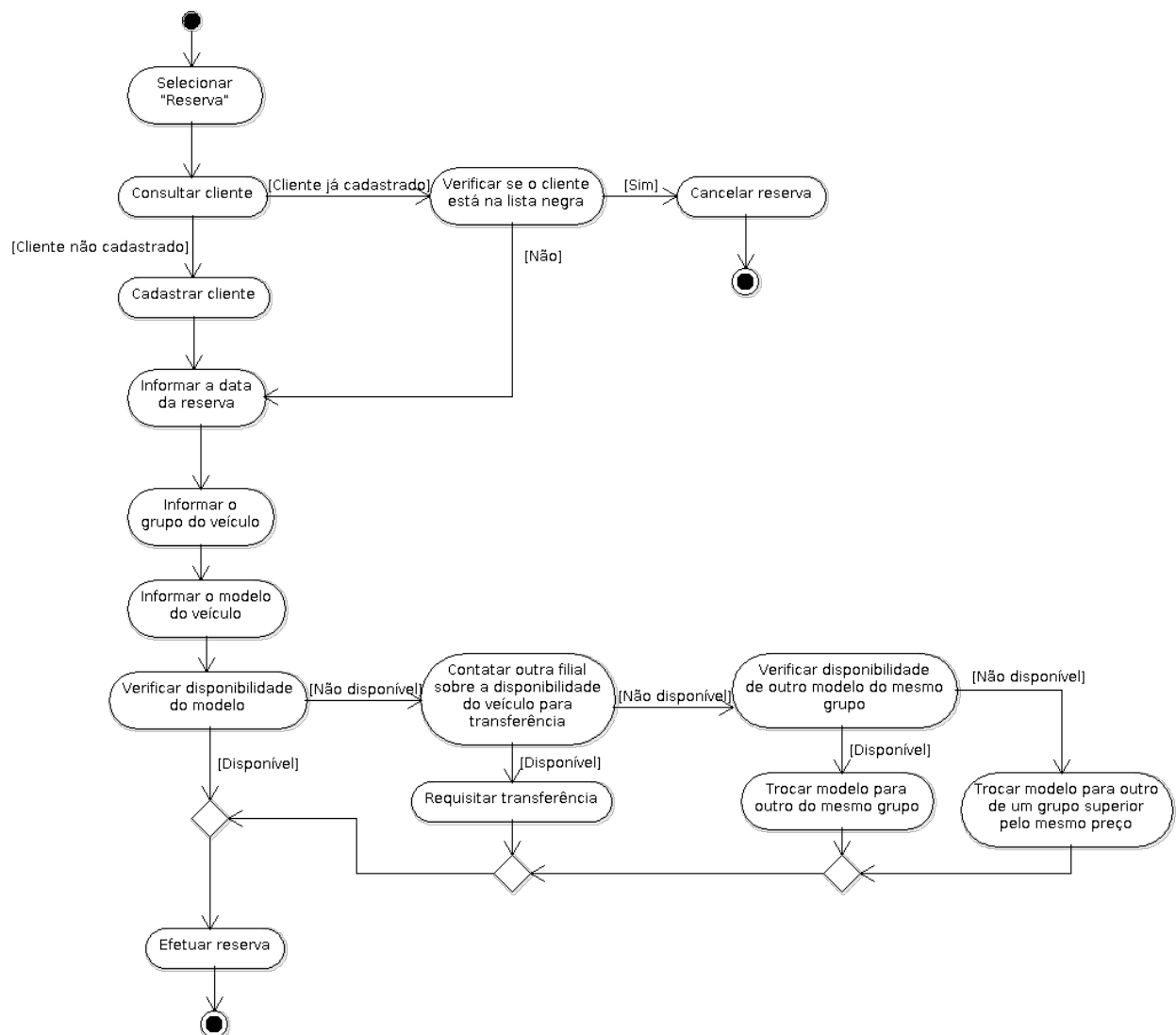
Um gerente pode cadastrar um novo carro no sistema, preenchendo todas as informações do automóvel no processo.



*Diagrama de sequência para o cadastro de automóveis*

### 3.1.3 - Reserva

Um agente de vendas ou gerente pode registrar uma reserva de um automóvel para um cliente. O cliente deve estar cadastrado para efetuar a reserva, o atendente deve efetuar o cadastro se necessário. Clientes que constam na lista negra do sistema não poderão efetuar reservas. Para efetuar a reserva o cliente deve informar o grupo do automóvel a ser reservado. Caso não haja um carro do modelo pedido na filial, este será trocado por outro do mesmo grupo. Também não havendo um automóvel do grupo pedido, o agente deverá entrar em contato com outra filial para requisitar uma transferência de carro. Em último caso, o cliente o grupo do automóvel será trocado para um grupo superior, mantendo-se o preço da reserva.



## Reservar

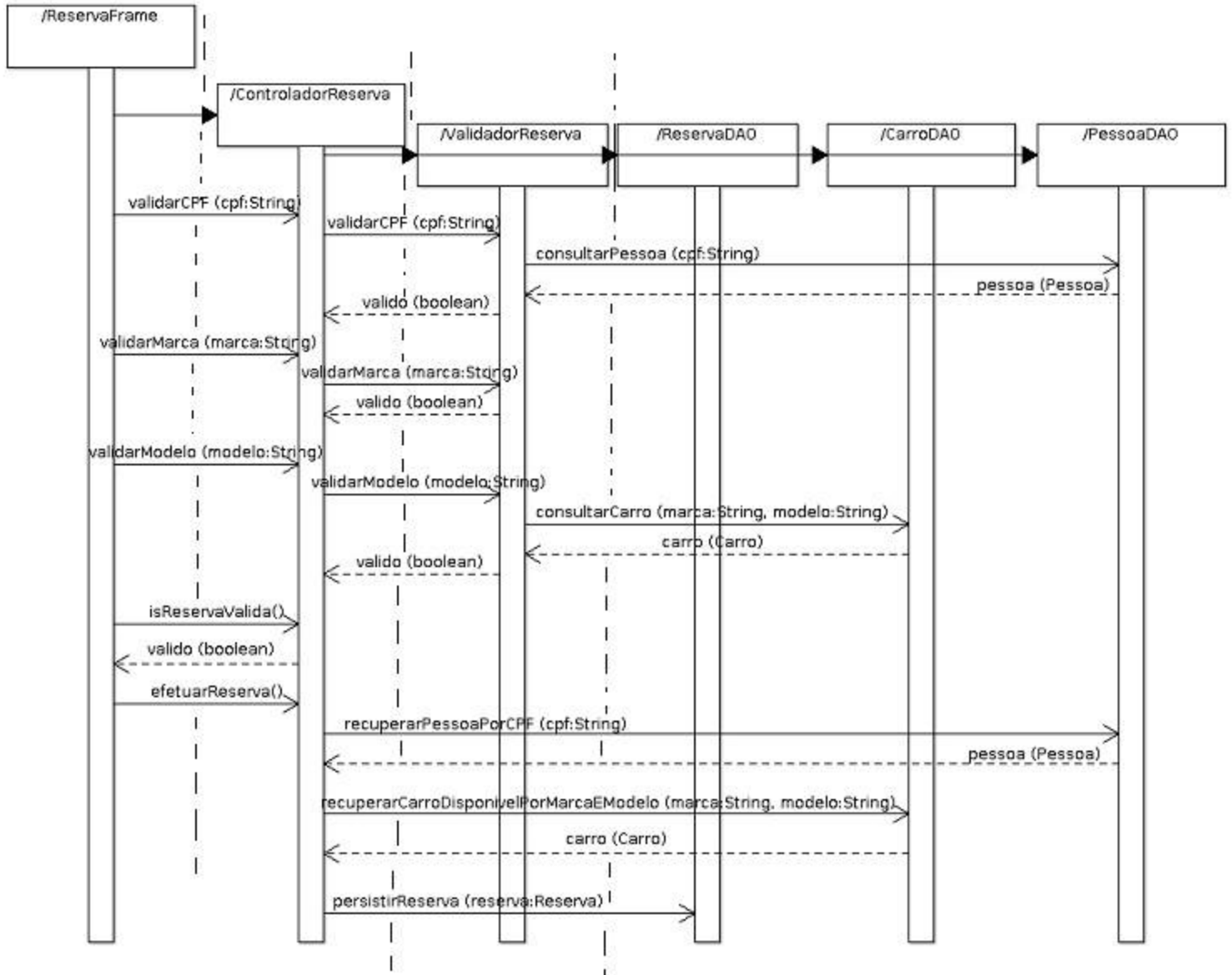
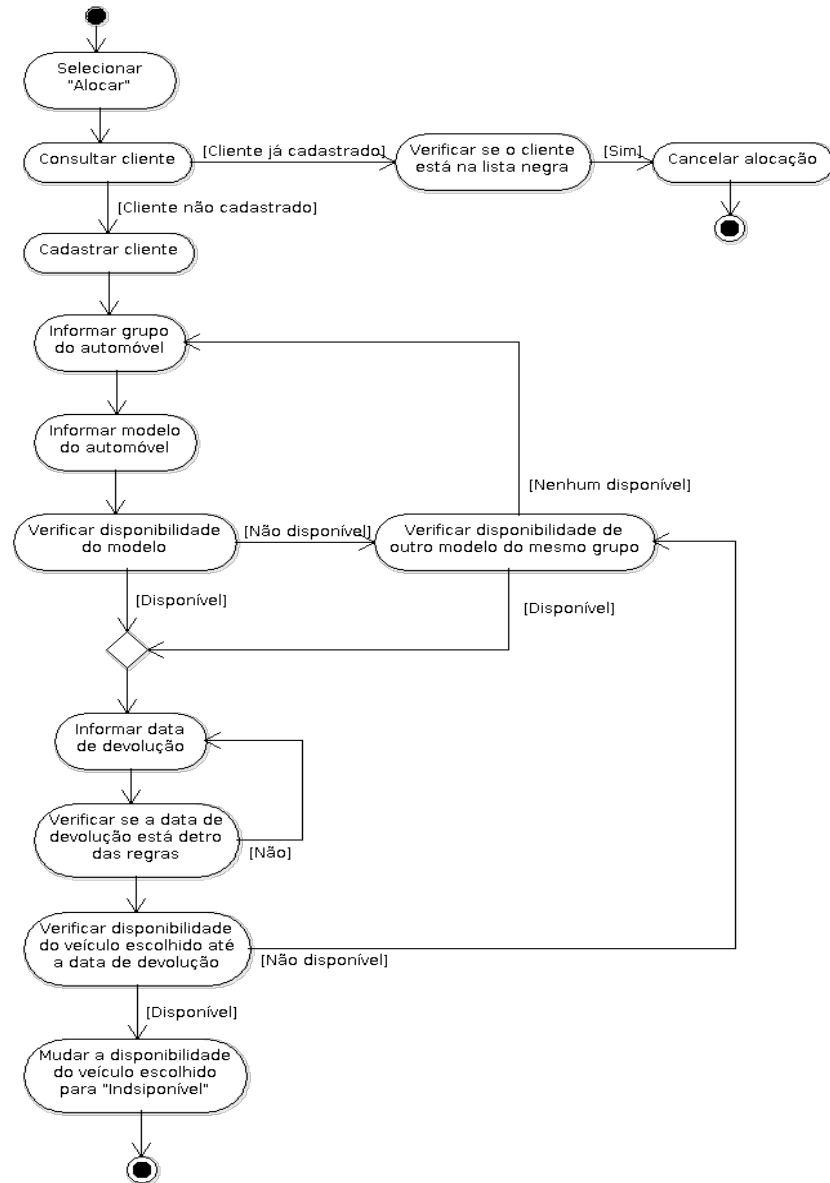


Diagrama de sequência da reserva



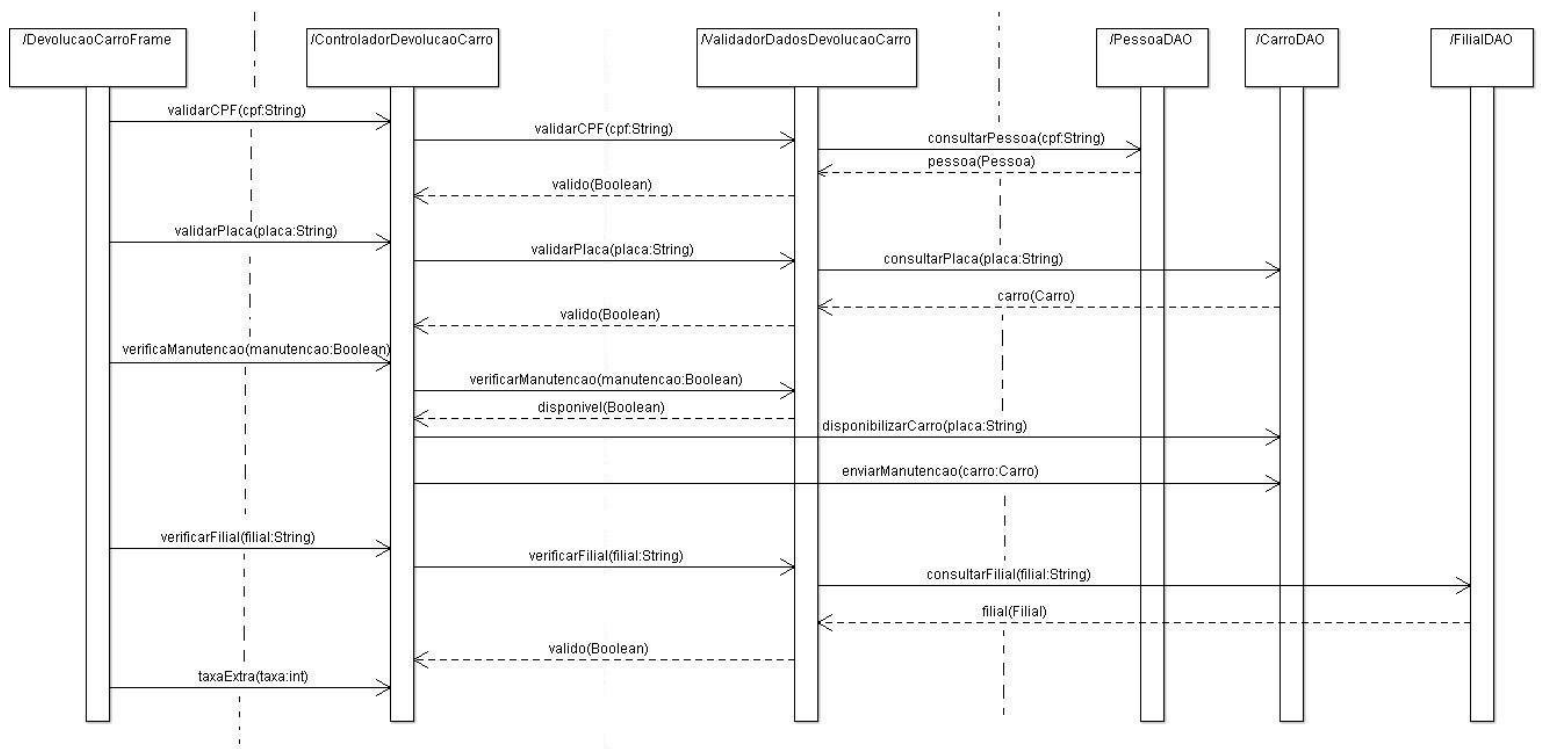
#### 3.1.4 - Alocação Imediata

Um atendente ou gerente pode executar a alocação imediata de um automóvel para um cliente. O cliente deve estar cadastrado para que a alocação seja efetuada, o atendente deve efetuar o cadastro se necessário. Alocações não podem ser feitas para clientes que estão na lista negra. O cliente deve informar o grupo e modelo do carro que ele pretende usar, assim como a data de devolução do automóvel, caso um carro deste modelo não esteja disponível para o período escolhido, o cliente pode mudar essas informações.



### 3.1.5 - Devolução

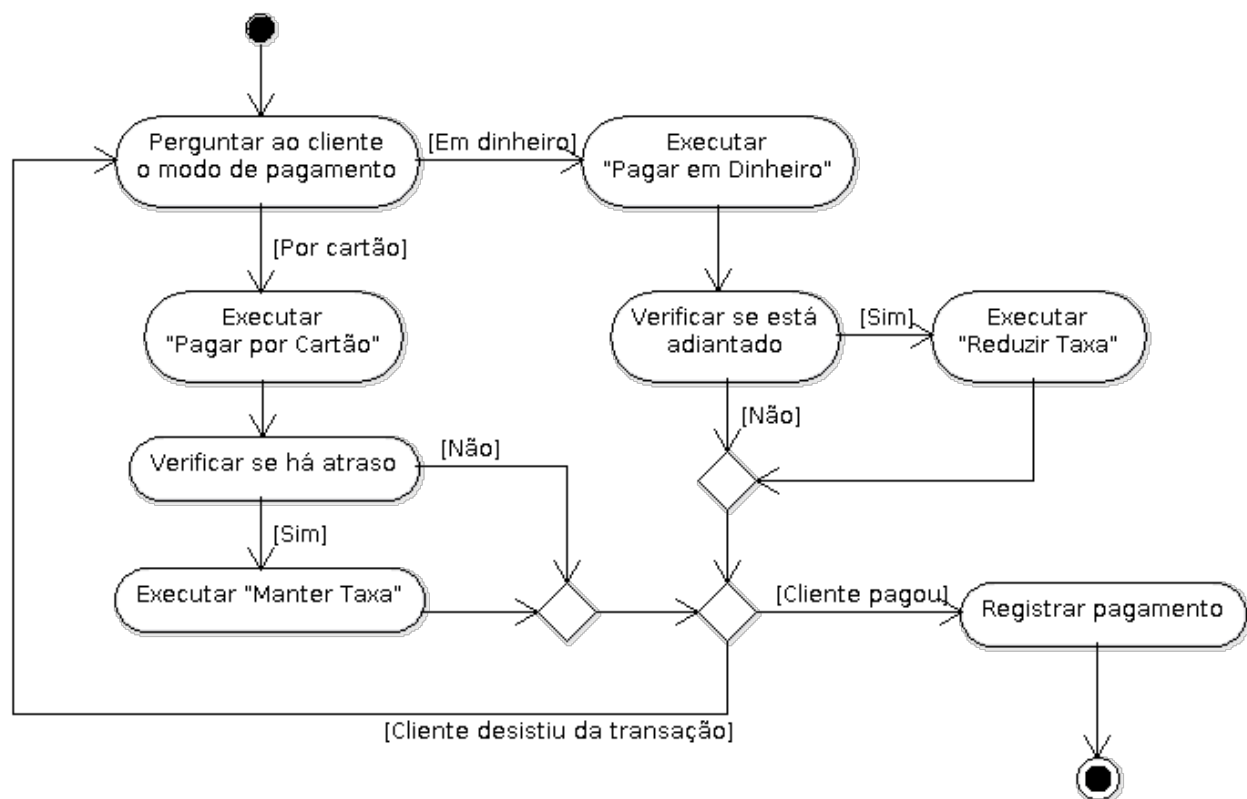
Ao fazer uma reserva ou alocação, o atendente ou gerente deve garantir que o automóvel seja devolvido a alguma filial do Você-Aluga. O cliente pode devolver o carro em outra filial da rede mediante ao pagamento de uma taxa adicional. Se o automóvel for devolvido a outra filial, a propriedade dele é transferida para a filial de devolução.



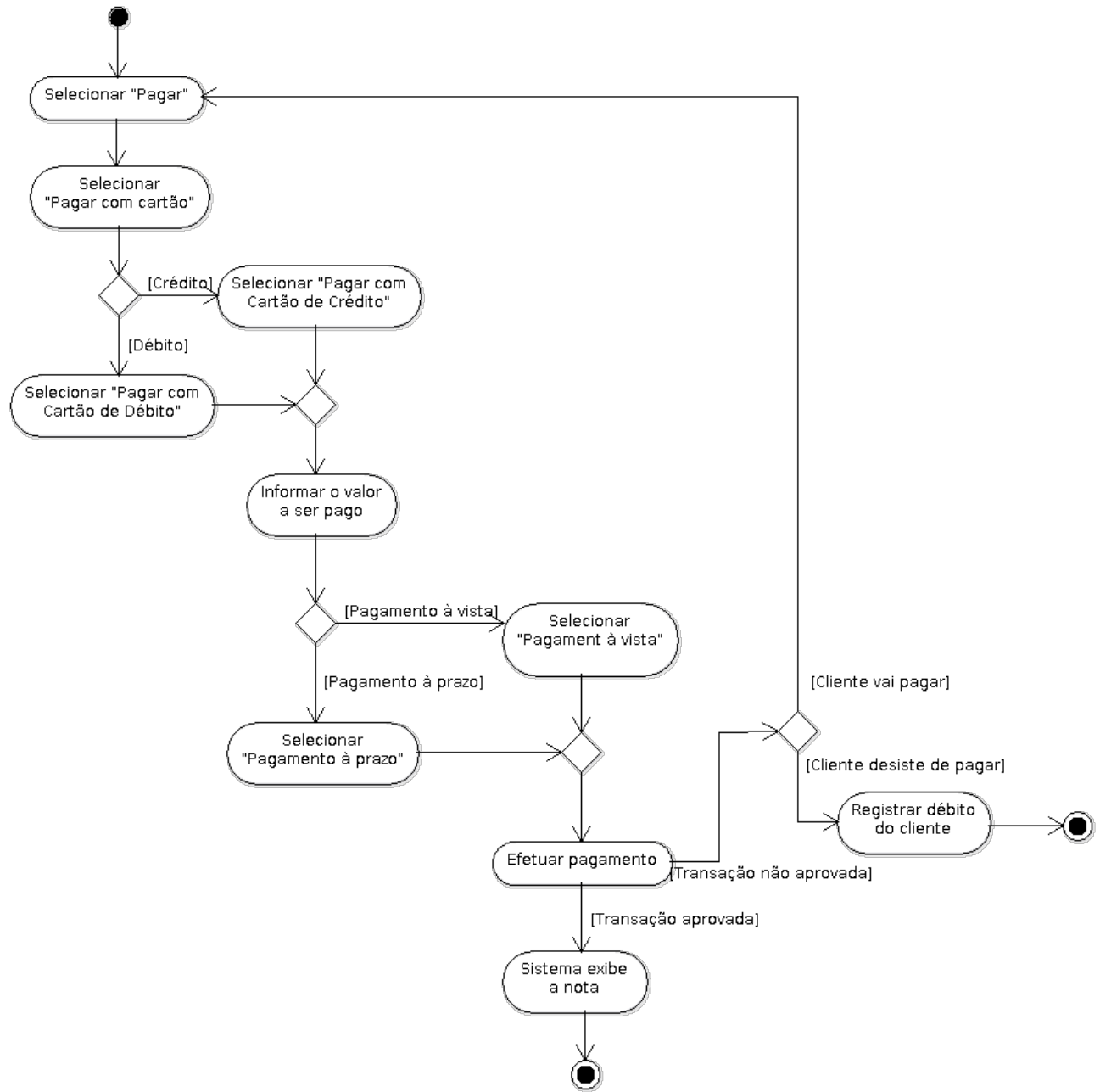
*Diagrama de sequência de devolução*

### 3.1.6 - Registro de Pagamento

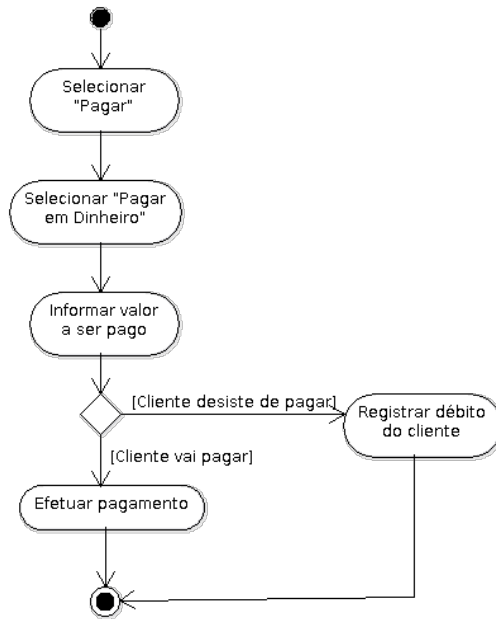
Um atendente ou gerente pode registrar o pagamento de um cliente por uma reserva ou alocação. O cliente deve ter a opção de pagar em dinheiro ou com cartão. Caso haja atraso no pagamento, o cliente deve pagar uma taxa de atraso. Caso o pagamento esteja sendo feito adiantado, haverá uma redução de taxa. O cliente deverá pagar uma taxa de danificação caso o veículo esteja sendo devolvido danificado e uma taxa de transferência caso esteja devolvendo o carro numa filial diferente da qual ele retirou o mesmo.



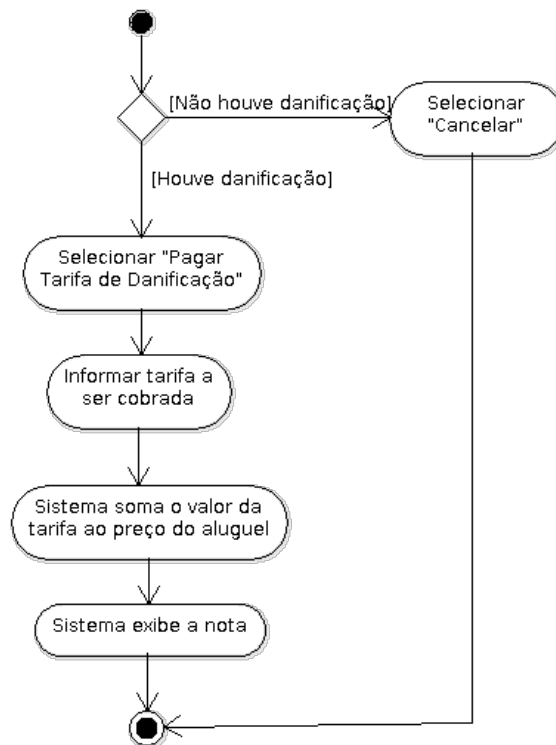
*Registrar Pagamento*



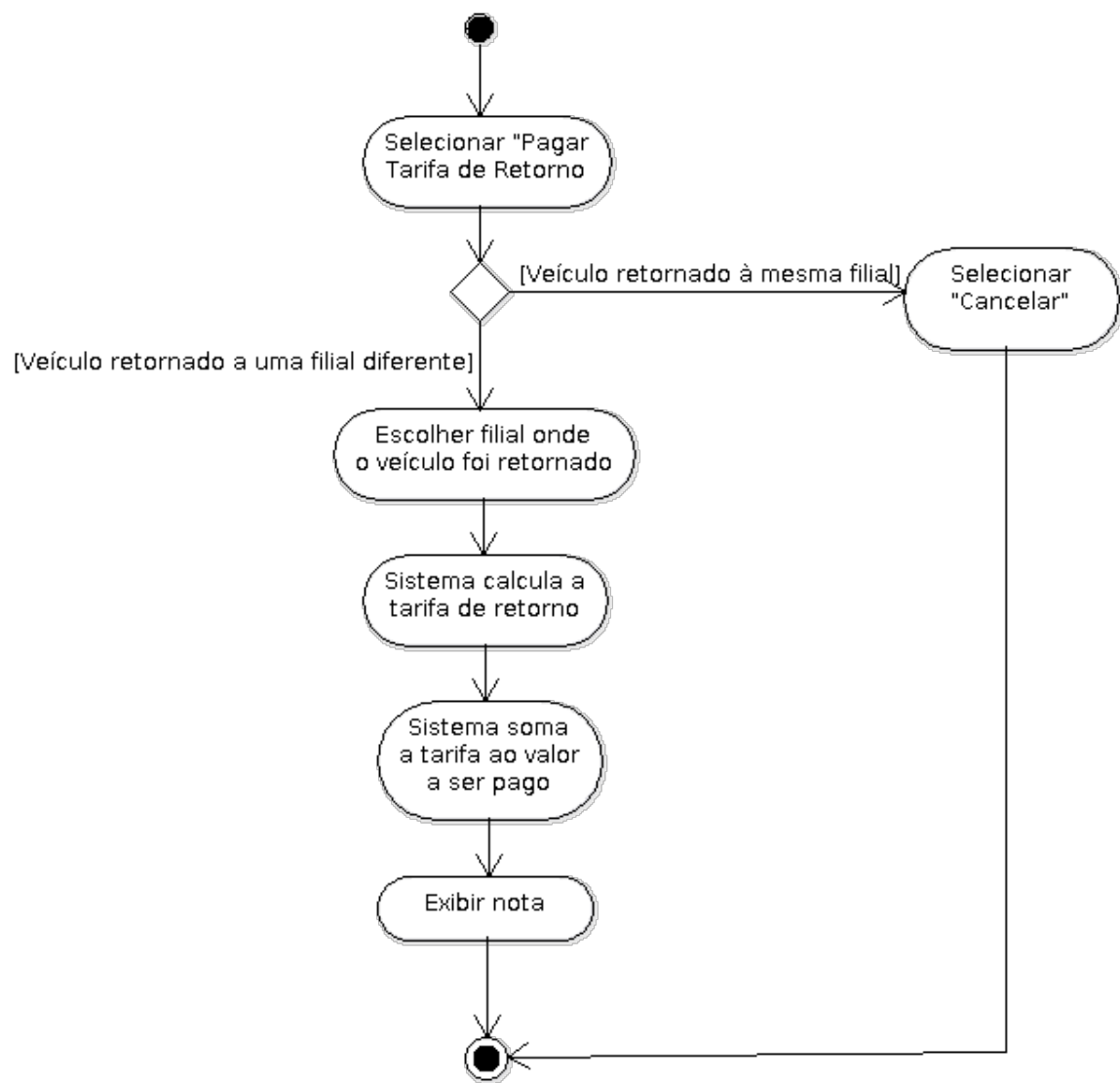
*Pagar com cartão*



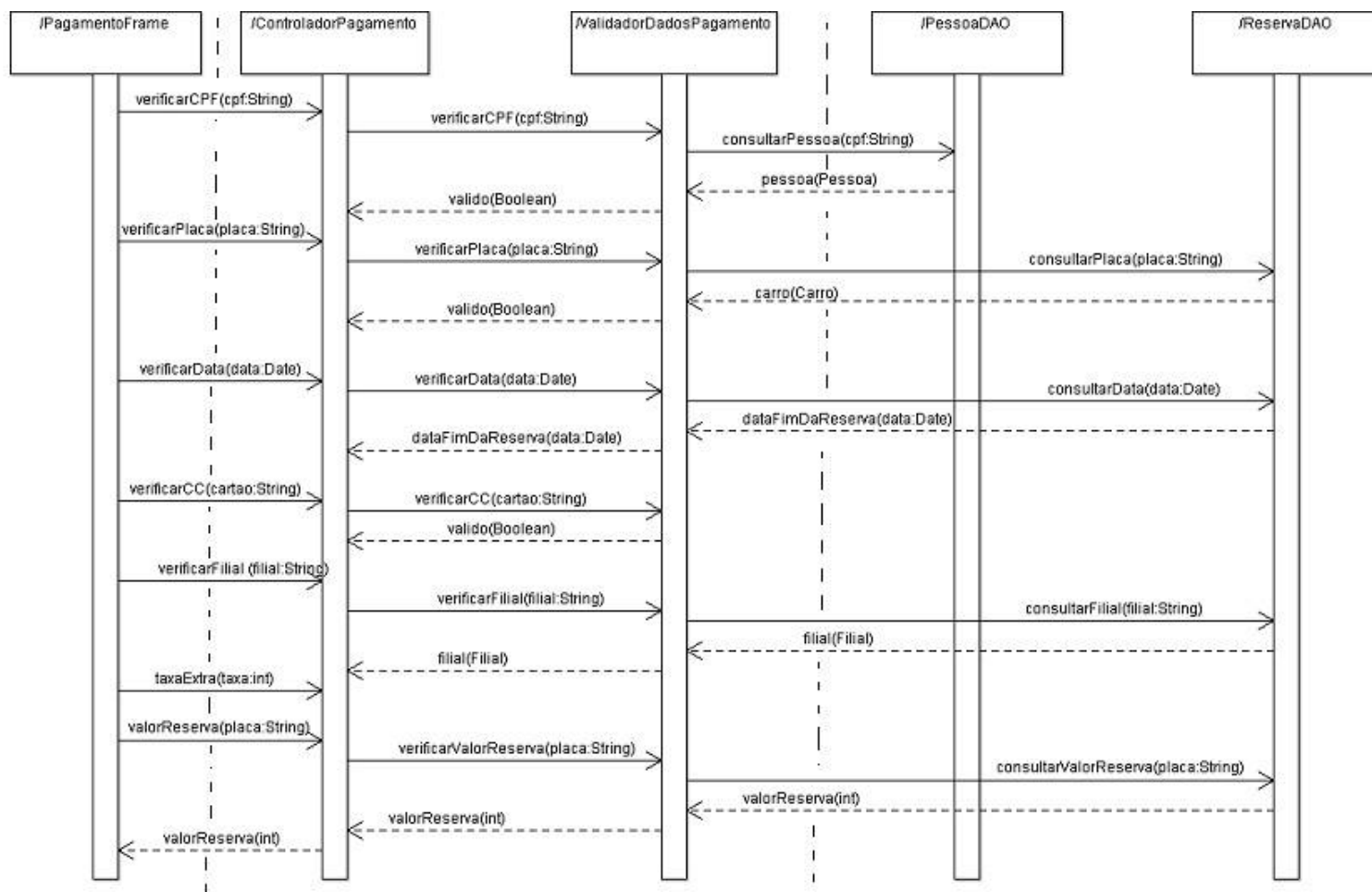
*Pagar em dinheiro*



### *Pagar tarifa de danificação*



### *Pagar tarifa de retorno*

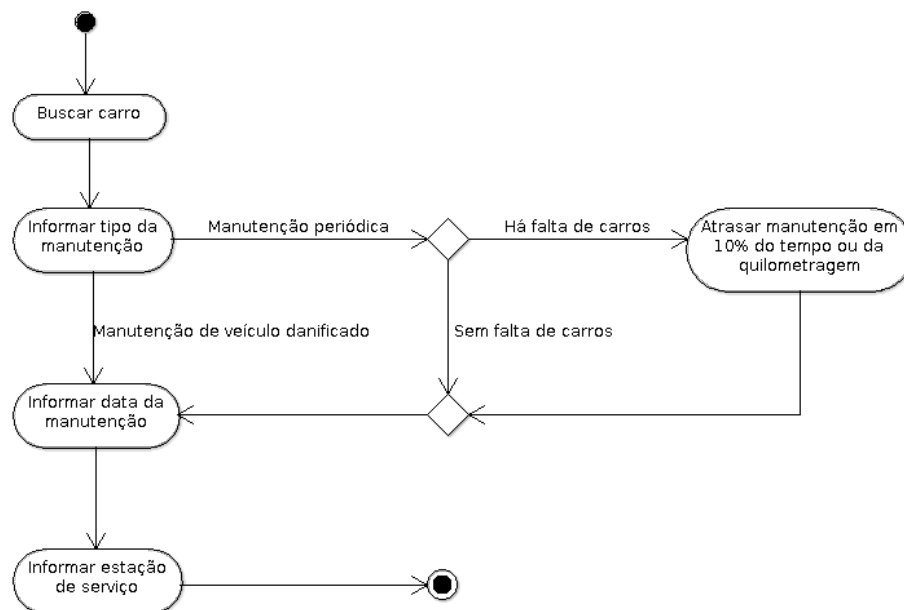


*Diagrama de sequência de pagamento*

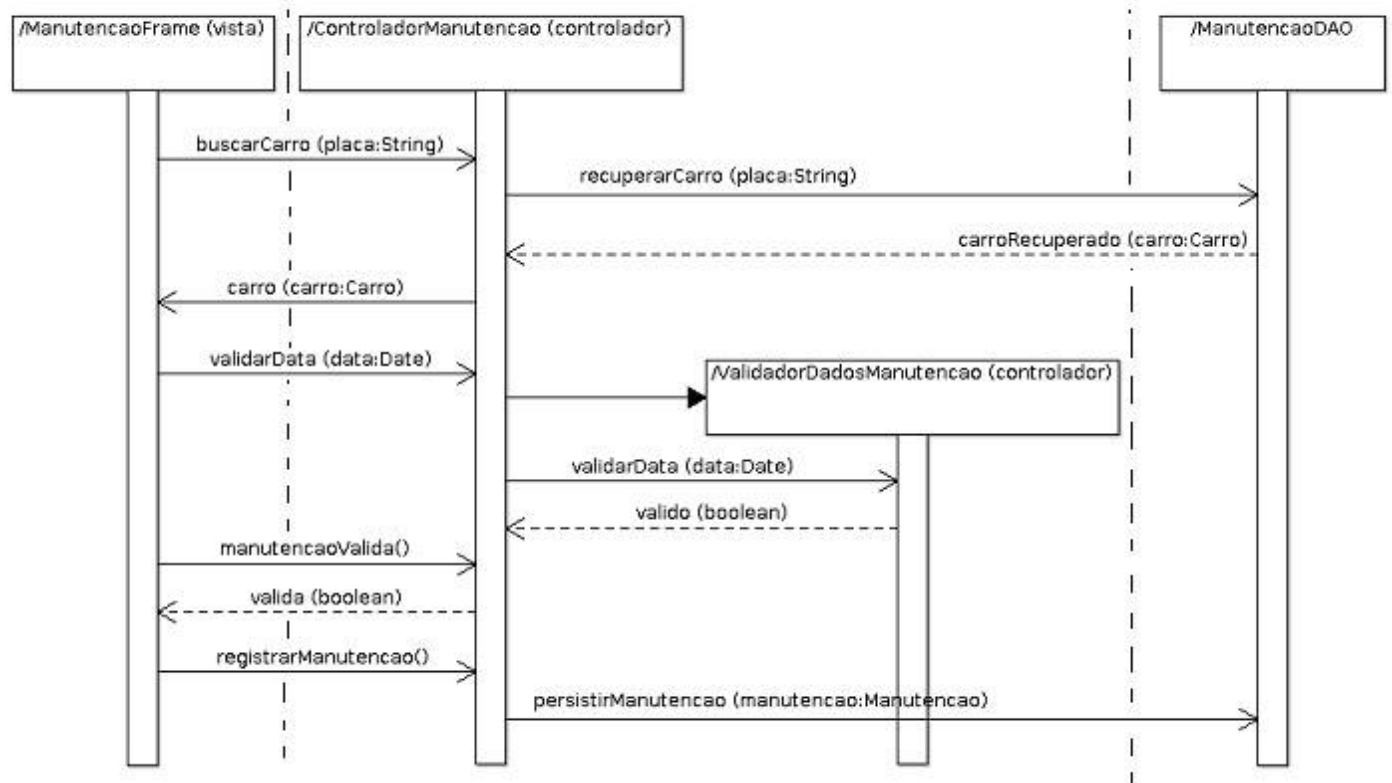
### 3.1.7 - Manutenção

Um gerente pode enviar um carro para a manutenção. Os carros devem ser enviados para a manutenção a cada 3 meses, 10 mil quilômetros rodados ou no caso de uma falha ocasional. Cada carro só pode ter uma manutenção agendada por vez, mas o serviço pode durar vários dias. No caso de uma manutenção periódica, a manutenção pode ser retardada por até 10% do tempo ou da quilometragem caso haja falta de carros disponíveis.



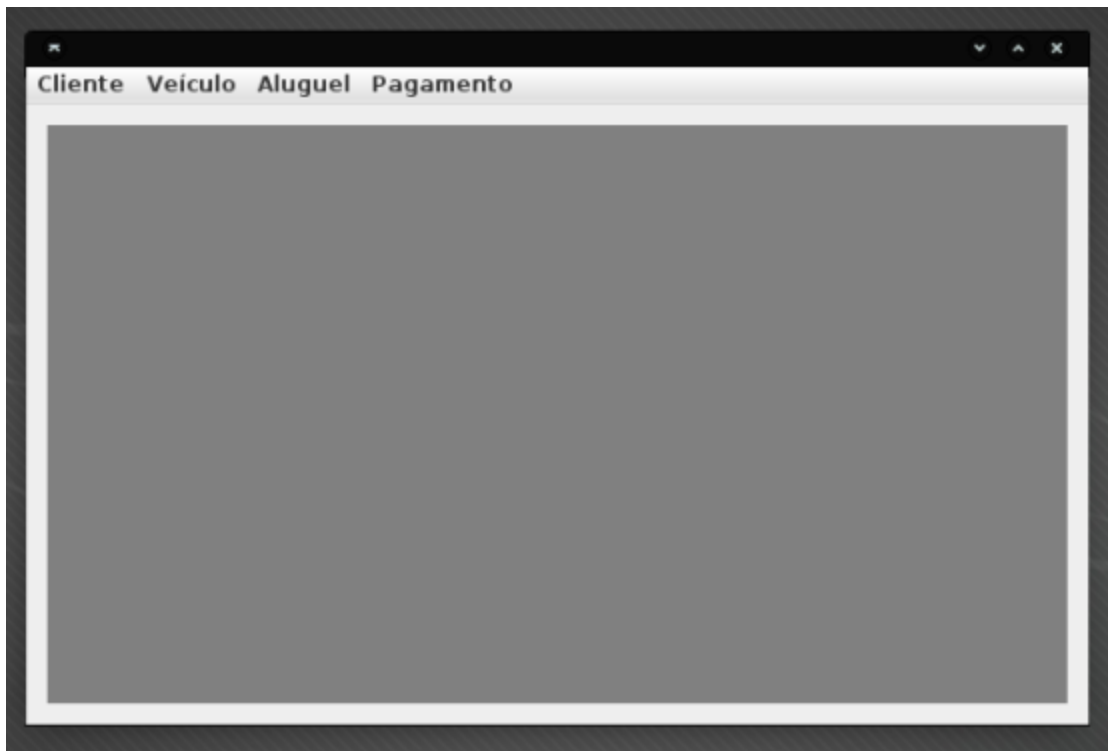


*Agendamento de manutenção*



*Diagrama de sequência do agendamento de manutenção*

### 3.2 Telas



*Tela Principal*

The screenshot shows a window titled "Cliente Veículo Aluguel Pagamento" with a tab labeled "Informações da Reserva". The window is divided into three main sections. The top-left section, titled "Informações do Cliente", contains the text: "Nome: Carlos", "CPF: 0809809", and "Carteira: 0897897 (Tipo A)". The bottom-left section, titled "Informações do Veículo", contains: "Marca: Fiat", "Modelo: Doblo", and "Diária: R\$100.00". The right section, titled "Informações da Reserva", contains: "Data de Início: 01/10/2014", "Data de Fim: 02/10/2014", "Valor: R\$100.00", and "Pago antecipadamente: Não". At the bottom of the right section is a button labeled "Alugar". At the bottom center of the window is a button labeled "Voltar".

Informações do Cliente	
Nome:	Carlos
CPF:	0809809
Carteira:	0897897 (Tipo A)

Informações do Veículo	
Marca:	Fiat
Modelo:	Doblo
Diária:	R\$100.00

Informações da Reserva	
Data de Início:	01/10/2014
Data de Fim:	02/10/2014
Valor:	R\$100.00
Pago antecipadamente:	Não

**Alugar**

**Voltar**

*Aluguel de veículo*

The screenshot shows a window titled "Cliente Veículo Aluguel Pagamento" with a tab labeled "Cadastro de Cliente". The window contains several input fields for client registration: "Nome" (a single-line text box), "CPF" (a single-line text box), "Carteira" (a single-line text box), "Categoria" (a dropdown menu with "A" selected), "RG" (a single-line text box), "Telefone" (a single-line text box), and "Email" (a single-line text box). At the bottom right of the window are two buttons: "Salvar" and "Cancelar".

Nome

CPF  Carteira  Categoria

RG  Telefone

Email

**Salvar** **Cancelar**

*Cadastro de Cliente*

Cliente Veículo Aluguel Pagamento

**Cadastro de Veículo**

Marca

Ano

Modelo

Placa

☐ Disponível

Preço

Diária

Última Manutenção

Novembro 2014

	Dom	Seg	Ter	Qua	Qui	Sex	Sáb
44							1
45	2	3	4	5	6	7	8
46	9	10	11	12	13	14	15
47	16	17	18	19	20	21	22
48	23	24	25	26	27	28	29
49	30						

**Salvar** **Cancelar**

*Cadastro de Veículo*

Cliente Veículo Aluguel Pagamento

☐ Venda de Veículo

PAGAMENTO COM DINHEIRO

Pagar

PAGAMENTO NO DÉBITO

Pagar

PAGAMENTO NO CRÉDITO

Parcelas

Pagar

*Compra de Veículo*

The screenshot shows a window titled "Cliente Veículo Aluguel Pagamento". Inside, there is a tab labeled "Consultar Cliente". Below the tab, there is a text input field for "CPF:" and a "Consultar" button. The results area displays a list of client information:

- CPF: 444444 | Nome: Pessoa Teste
- CPF: 876876 | Nome: Sildenir
- CPF: 0809809 | Nome: Carlos
- CPF: 99988 | Nome: Diego
- CPF: 123123 | Nome: Carlos Gordo

At the bottom of the window, there are two buttons: "Selecionar" and "Voltar".

*Consulta de Cliente*

The screenshot shows a window titled "Cliente Veículo Aluguel Pagamento". Inside, there is a tab labeled "Consultar Reserva". Below the tab, there is a text input field for "CPF do Cliente:" and a "Consultar" button. The results area displays a list of reservation information:

- Carro: Ford Ka | Cliente: Carlos Gordo | Duração: 28/10/2014 a 29/10/2014
- Carro: Marca Teste Teste | Cliente: Pessoa Teste | Duração: 28/10/2014 a 28/10/2014
- Carro: Marca Teste Teste | Cliente: Pessoa Teste | Duração: 28/10/2014 a 28/10/2014
- Carro: Fiat Doblo | Cliente: Carlos | Duração: 01/10/2014 a 02/10/2014
- Carro: Renault Sandero | Cliente: Diego | Duração: 28/10/2014 a 29/10/2014
- Carro: Teste Teste | Cliente: Sildenir | Duração: 03/11/2014 a 06/11/2014
- Carro: Marca Teste Teste | Cliente: Pessoa Teste | Duração: 28/10/2014 a 28/10/2014
- Carro: Teste Teste | Cliente: Sildenir | Duração: 04/11/2014 a 07/11/2014
- Carro: Fiat Doblo | Cliente: Carlos Gordo | Duração: 10/11/2014 a 14/11/2014

At the bottom of the window, there are two buttons: "Selecionar" and "Voltar".

*Consulta de Reserva*

The screenshot shows a window titled 'Cliente Veículo Aluguel Pagamento'. The active tab is 'Consultar Veículo'. It features a search form with a 'Modelo:' label, a text input field, a checkbox labeled 'Só disponíveis', and a 'Consultar' button. Below the form is a list of vehicle records, each showing the model, plate, and availability status. At the bottom are 'Selecionar' and 'Voltar' buttons.

Modelo	Placa	Status
Modelo: adasd adasd	Placa: daadasd	Indisponível
Modelo: Ford Ka	Placa: PLC-4567	Indisponível
Modelo: Teste Teste	Placa: TES-1234	Disponível
Modelo: Ford Fusion	Placa: GJK-8754	Disponível
Modelo: Marca Teste Teste	Placa: TST-1234	Indisponível
Modelo: Fiat Uno	Placa: YUT-8678	Indisponível
Modelo: Renault Sandero	Placa: XYZ-9876	Indisponível
Modelo: Ford Fusion	Placa: HJK-4566	Indisponível
Modelo: Fiat Doblo	Placa: ABC-1234	Disponível
Modelo: Ford Ka	Placa: DGD-5676	Indisponível

*Consulta de Veículo*

The screenshot shows a window titled 'Cliente Veículo Aluguel Pagamento'. The active tab is 'Informações do Cliente'. It displays the following information: Name (Pessoa Teste), CPF (444444) and RG (23423), License (23424) and Category (A), Phone (44444444), Email (pessoa@teste.com), and Blacklist status (Sim, highlighted in red). A 'Voltar' button is at the bottom.

**Nome: Pessoa Teste**

CPF: 444444    RG: 23423

Carteira: 23424    Categoria: A

Telefone: 44444444

Email: pessoa@teste.com

Lista negra: **Sim**

*Informações Cliente*



Cliente Veículo Aluguel Pagamento

**Informações da Reserva**

Informações do Cliente	Informações da Reserva
Nome: Carlos Gordo	Data de Inicio: 10/11/2014
CPF: 123123	Data de Fim: 14/11/2014
Carteira: 1231231 (Tipo A)	Valor: R\$400.00
	Pago antecipadamente: Não
	<input type="button" value="Cancelar"/>

*Informações de Reserva com Botão de Cancelamento*

Cliente Veículo Aluguel Pagamento

**Informações da Reserva**

Informações do Cliente	Informações da Reserva
Nome: Carlos	Data de Inicio: 01/10/2014
CPF: 0809809	Data de Fim: 02/10/2014
Carteira: 0897897 (Tipo A)	Valor: R\$100.00
	Pago antecipadamente: Não

*Informações de Reserva*

Cliente Veículo Aluguel Pagamento

Informações do Veículo

**Veículo: Ford Fusion**

Marca: Ford    Modelo: Fusion  
Ano: 2014  
Placa: GKJ-8754  
Última manutenção: 24/09/2014

Diária: R\$300.00  
Preço: R\$6000.00    Vendido: Não   

Disponível: **Sim**

*Informações Veículo*

Cliente Veículo Aluguel Pagamento

Reserva Por Modelo



Dados Pessoais

CPF

Dados Veículo

Marca     Modelo

Duração da Reserva

Data Inicio      Data Fim  

*Reserva de Veículo*

### 3.3 Códigos

Abaixo apresentamos alguns screenshots dos códigos implementados, todos os codigos podem ser encontrados dentro do arquivo zip em que se encontra esse arquivo.



```
ValidadorDadosCadastroCliente.java
public class ValidadorDadosCadastroCliente
{
    private static final int TAMANHO_MAXIMO_NOME = 3;

    public boolean validarNome(String nome)
    {
        if(nome == null)
            return false;

        if((nome.length() < TAMANHO_MAXIMO_NOME) || !nome.matches("[A-Z][a-z]+([A-Z][a-z]+)?"))
            return false;
        else
            return true;
    }

    public boolean validarCpf(String cpf)
    {
        boolean valido = false;

        if(cpf == null)
            return false;

        if(cpf.matches("[0-9]+" && (cpf.length() <= 11)))
            valido = true;

        return valido;
    }

    public boolean validarRG(String rg)
    {
        if(rg == null)
```



```
ControladorInformacoesVeiculo.java
{
    public void atualizarManutencao()
    {
        veiculo.setDisponivel("false");
        veiculo.setUltimaManutencao(new Date());
        atualizarVeiculo();
        recuperarFilial();
        popularManutencao();
        manutencaoDAO.persistirManutencao(manutencao);
        controladorConsultaVeiculo.prepopular();
    }

    public void removerManutencao()
    {
        veiculo.setDisponivel("true");
        atualizarVeiculo();
        manutencaoDAO = new ManutencaoDAO();
        manutencaoDAO.removerManutencao(veiculo.getCarroOid());
        controladorConsultaVeiculo.prepopular();
    }

    private void popularManutencao()
    {
        manutencaoDAO = new ManutencaoDAO();
    }
}
```

```
ControladorConsultaReserva.java
{
    public void atualizarListadeReservas()
    {
        reservaDAO.refresh("Reserva");
        reservas.clear();
        prepopular();
    }

    public void filtrarReservas(String cpf)
    {
        reservas.clear();
        if (cpf == null || cpf.equals(""))
        {
            reservas.addAll(reservaDAO.recuperarTodasReservas());
        }
        else
        {
            reservas.addAll(reservaDAO.recuperarReservasPorCpfCliente(cpf));
        }
        popularTextoReservas();
    }

    private void popularTextoReservas()
    {
        textoReservas.clear();
        for (Reserva r : reservas)
        {
            textoReservas.add(formatarReserva(r));
        }
    }
}
```

```
CarroDAO.java PessoaDAO.java ControladorCadastroCliente.java
public class ControladorCadastroCliente
{
    private ValidadorDadosCadastroCliente validadorDadosCadastro = new ValidadorDadosCadastroCliente();
    private PessoaDAO pessoaDAO = new PessoaDAO();
    private boolean cadastroValido = true;
    private ArrayList<String> mensagensCadastro;

    public ControladorCadastroCliente()
    {
        pessoaDAO = new PessoaDAO();
        cadastroValido = true;
        mensagensCadastro = new ArrayList<String>();
    }

    public void validarNome(String nome)
    {
        if (!validadorDadosCadastro.validarNome(nome))
        {
            mensagensCadastro.add("Nome deve mais de 3 letras");
            cadastroValido = false;
        }
    }

    public void validarCpf(String cpf)
    {
        if (!verificarCPF(cpf))
        {

```

```
ControladorAluguelVeiculoPorReserva.java
public class ControladorAluguelVeiculoPorReserva {
    private Aluguel aluguel;
    private Reserva reserva;

    public ControladorAluguelVeiculoPorReserva(Reserva reserva)
    {
        this.reserva = reserva;
    }

    public boolean reservaPodeSerAlugada()
    {
        if (reserva.getDataFim().before(new Date()) && reserva.getCarro().getDisponivel().equals(true) )
        {
            return false;
        }
        else
        {
            return true;
        }
    }

    public void alugar()
    {
        if (!reservaPodeSerAlugada())
        {
            return;
        }

        AluguelDAO aluguelDAO = new AluguelDAO();
        aluguel = new Aluguel(reserva.getPessoa(), reserva.getCarro(), reserva, reserva.getDataInicio(), reserva.getDataFim(), reserva.isPagoAntecipado());
    }
}
```

```
CarroDAO.java

}

public Carro recuperarCarroDisponivelPorMarcaEModelo(String marca, String modelo)
{
    String query = "select * from Carro c where c.marca = :marca and c.modelo = :modelo and c.disponivel = 'true'";
    Carro veiculo = (Carro) entityManager.createNativeQuery(query, Carro.class).setParameter("marca", marca).setParameter("modelo", modelo).getSingleResult();
    return veiculo;
}

public List<Carro> recuperarTodosCarros()
{
    List<Carro> resultado;

    TypedQuery<Carro> query = entityManager.createQuery("select c from Carro c", Carro.class);
    resultado = query.getResultList();

    return resultado;
}

public List<Carro> recuperarCarrosDisponiveis()
{
    List<Carro> resultado;

    TypedQuery<Carro> query = entityManager.createQuery("select c from Carro c where c.disponivel = 'true'", Carro.class);
    resultado = query.getResultList();

    return resultado;
}

public List<Carro> recuperarCarrosPorModelo(String modelo)
```

```
AbstractDAO.java

package br.model;

import java.util.UUID;

public class AbstractDAO
{
    public EntityManagerFactory entityManagerFactory;
    public EntityManager entityManager;

    public void criarEntityManager(String entidade)
    {
        entityManagerFactory = Persistence.createEntityManagerFactory(entidade);
        entityManager = entityManagerFactory.createEntityManager();
    }

    public String criarOid()
    {
        UUID oid = UUID.randomUUID();

        return oid.toString();
    }

    public void refresh(String entidade)
    {
        entityManager.close();
        criarEntityManager(entidade);
    }
}
```

### **3.4 Banco de Dados**

O modelo de entidade-relacionamento pode ser encontrado na sessão 2.3.3

### **3.4 Modelo MVC**

Os detalhes do modelo MVC podem ser encontrados na sessão 2.1.1 e o diagrama da arquitetura mvc pode ser encontrado na sessão 2.3.4

## **4. Sobre os Testes aplicados**

### **4.1 - Testes para cadastro de cliente**

#### **4.1.1 - Testes Unitários**

##### **Validador: testValidarNome\_NomeComMaisDe3Letras\_DeveRetornarTrue**

Passa para o validador de cadastro de cliente uma entrada com um nome válido, sem números e com mais de 3 letras e espera que a entrada seja aceita.

##### **Validador: testValidarNome\_NomeComMenosDe3Letras\_DeveRetornarFalse**

Passa para o validador de cadastro de cliente uma entrada com um nome com menos de 3 letras e espera que a entrada seja rejeitada.



**Validador: testValidarNome\_NomeNaoPodeTerNumeros\_DeveRetornarFalse**

Passa para o validador de cadastro de cliente uma entrada com um nome que contém números e espera que a entrada seja rejeitada.

**Validador: testValidarNome\_NomeNulo\_DeveRetornarFalse**

Passa para o validador de cadastro de cliente uma entrada com um nome nulo (vazio) e espera que a entrada seja rejeitada.

**Validador: testValidarCPF\_CPFNulo\_DeveRetornarFalse**

Passa para o validador de cadastro de cliente uma entrada com um CPF nulo (vazio) e espera que a entrada seja rejeitada.

**Validador: testValidarCPF\_CPFComLetras\_DeveRetornarFalse**

Passa para o validador de cadastro de cliente uma entrada com um CPF que contém letras e espera que a entrada seja rejeitada.

**Validador: testValidarCPF\_CPFSemLetras\_DeveRetornarTrue**

Passa para o validador de cadastro de cliente uma entrada com um CPF comum, com 10 caracteres, sem letras, e espera que a entrada seja aceita.

**Validador: testValidarCPF\_CPFComMaisDe11Caracteres\_DeveRetornarFalse**

Passa para o validador de cadastro de cliente uma entrada com um CPF com mais de 11 caracteres e espera que a entrada seja rejeitada.

**Validador: testValidarRG\_RGSemLetras\_DeveRetornarTrue**

Passa para o validador de cadastro de cliente uma entrada com um RG comum, sem letras, e espera que a entrada seja aceita.

**Validador: testValidarRG\_RGComLetras\_DeveRetornarFalse**

Passa para o validador de cadastro de cliente uma entrada com um RG com letras e espera que a entrada seja rejeitada.

**Validador: testValidarRG\_RGNulo\_DeveRetornarFalse**

Passa para o validador de cadastro de cliente uma entrada com um RG nulo (vazio) e espera que a entrada seja rejeitada.

**Validador: testValidarTelefone\_TelefoneSemLetras\_DeveRetornarTrue**

Passa para o validador de cadastro de cliente uma entrada com um telefone comum, sem letras, e espera que a entrada seja aceita.

**Validador: testValidarTelefone\_TelefoneComLetras\_DeveRetornarFalse**

Passa para o validador de cadastro de cliente uma entrada com um telefone que contém letras e espera que a entrada seja rejeitada.

**Validador: testValidarTelefone\_TelefoneNulo\_DeveRetornarFalse**

Passa para o validador de cadastro de cliente uma entrada com um telefone nulo (vazio) e espera que a entrada seja rejeitada.

**Validador:****testValidarTelefone\_TelefoneComMaisDe8Caracteres\_DeveRetornarFalse**

Passa para o validador de cadastro de cliente uma entrada com um telefone com mais de 8 caracteres e espera que a entrada seja rejeitada.

**Validador: testValidarEmail\_EmailCorreto\_DeveRetornarTrue**

Passa para o validador de cadastro de cliente uma entrada com um email dentro do formato adequado e espera que a entrada seja aceita.

**Validador: testValidarEmail\_EmailInvalido\_DeveRetornarFalse**

Passa para o validador de cadastro de cliente uma entrada com um email fora do formato adequado e espera que a entrada seja rejeitada.

**Validador: testValidarCarteira\_CarteiraSemLetras\_DeveRetornarTrue**

Passa para o validador de cadastro de cliente uma entrada com uma carteira de motorista comum, sem letras, e espera que a entrada seja aceita.

**Validador: testValidarCarteira\_CarteiraComLetras\_DeveRetornarFalse**

Passa para o validador de cadastro de cliente uma entrada com uma carteira de motorista que contém letras e espera que a entrada seja rejeitada.

**Validador: testValidarCarteira\_CarteiraNulo\_DeveRetornarFalse**

Passa para o validador de cadastro de cliente uma entrada com uma carteira de motorista nula (vazia) e espera que a entrada seja rejeitada.

**4.1.2 - Testes Funcionais****Cadastro:****testCadastrarCliente\_CadastrarComSucesso\_ClienteDeveEstarNoBanco**

Envia ao validador dados válidos para o cadastro de um cliente e chama o método de cadastrar cliente. Concluída a operação, checa se o cliente consta no banco. Deve achar o cliente recém-cadastrado no banco.

**Cadastro: testVerificarCPF\_CPFNaoExisteNoBanco\_DeveRetornarTrue**

Chama o método que verifica se um CPF não consta no banco passando um CPF que não consta no banco. Espera que o método retorne verdadeiro.

**Cadastro: testVerificarCPF\_CPFExisteNoBanco\_DeveRetornarFalse**

Chama o método que verifica se um CPF não consta no banco passando um CPF que já consta no banco. Espera que o método retorne falso.

**Cadastro:**

**testValidarCPF\_CPFExisteNoBanco\_DeveInserirUmaMensagemNoBundle**

Passa para o validador um CPF que já consta no banco e chama o método que verifica se um CPF não consta no banco. Espera que ao final, uma mensagem tenha sido adicionada a lista de mensagens dizendo que o CPF já consta no banco.

**Cadastro:**

**testValidarCPF\_CPFNaoExisteNoBanco\_NaoDeveInserirUmaMensagemNoBundle**

Passa para o validador um CPF que não consta no banco e chama o método que verifica se um CPF não consta no banco. Espera que ao final, nenhuma mensagem tenha sido adicionada a lista de mensagens do sistema.

## **4.2 - Testes para cadastro de carro**

### **4.2.1 - Testes Unitários**

**Validador:**

**testValidadorDadosCadastroVeiculo\_ValidarModelo\_ModeloNaoNulo\_DeveRetornarTrue**

Passa para o validador de cadastro de carro uma entrada com um modelo válido e espera que a entrada seja aceita.

**Validador:**

**testValidadorDadosCadastroVeiculo\_ValidarModelo\_ModeloNulo\_DeveRetornarFalse**

Passa para o validador de cadastro de carro uma entrada com um modelo nulo (vazio) e espera que a entrada seja rejeitada.

**Validador:**

**testValidadorDadosCadastroVeiculo\_ValidarMarca\_MarcaNaoNula\_DeveRetornar True**

Passa para o validador de cadastro de carro uma entrada com uma marca válida e espera que a entrada seja aceita.

**Validador:**

**testValidadorDadosCadastroVeiculo\_ValidarMarca\_MarcaNula\_DeveRetornarFalse**

Passa para o validador de cadastro de carro uma entrada com uma marca nula e espera que a entrada seja rejeitada.

.

**Validador:**

**testValidadorDadosCadastroVeiculo\_ValidarPlaca\_PlacaNaoNula\_DeveRetornarTrue**

Passa para o validador de cadastro de carro uma entrada com uma placa comum, válida, e espera que a entrada seja aceita.

**Validador:**

**testValidadorDadosCadastroVeiculo\_ValidarPlaca\_PlacaNula\_DeveRetornarFalse**

Passa para o validador de cadastro de carro uma entrada com uma placa nula e espera que a entrada seja rejeitada.

**Validador:**

**testValidadorDadosCadastroVeiculo\_ValidarDataUltimaManutencao\_DataNoFuturo\_DeveRetornarFalse**

Passa para o validador de cadastro de carro uma entrada com uma data de última manutenção do carro que seja no futuro e espera que a entrada seja rejeitada.

**Validador:**

**testValidadorDadosCadastroVeiculo\_ValidarDataUltimaManutencao\_DataValida\_DeveRetornarTrue**

Passa para o validador de cadastro de carro uma entrada com uma data de última manutenção do carro válida, no passado, e espera que a entrada seja aceita.

**Controlador:**

**testControladorCadastroVeiculo\_DadosNaoNulos\_CadastroValidoDeveSerTrue**

Passa para o controlador de cadastro de carro uma entrada válida, chama o método de validação e espera que, ao final, o campo que indica se o cadastro é válido seja verdadeiro.

**Controlador:**

**testControladorCadastroVeiculo\_MarcaNula\_CadastroValidoDeveSerFalse**

Passa para o controlador de cadastro de carro uma entrada com o campo marca nulo, chama o método de validação e espera que, ao final, o campo que indica se o cadastro é válido seja falso.

**Controlador:**

**testControladorCadastroVeiculo\_ModeloNula\_CadastroValidoDeveSerFalse**

Passa para o controlador de cadastro de carro uma entrada com o campo modelo nulo, chama o método de validação e espera que, ao final, o campo que indica se o cadastro é válido seja falso.

**Controlador:**

**testControladorCadastroVeiculo\_PlacaNula\_CadastroValidoDeveSerFalse**

Passa para o controlador de cadastro de carro uma entrada com o campo placa nulo, chama o método de validação e espera que, ao final, o campo que indica se o cadastro é válido seja falso.

#### **4.2.1 - Testes Funcionais**

##### **testCadastrarVeiculo\_DeveCadastrarComSucesso**

Passa para o controlador de cadastro de veículo dados válidos, chama o método de cadastrar, depois verifica se o carro foi cadastrado com sucesso.

##### **testCadastrarVeiculo\_ValidarPlaca\_PlacaJaCadastrada\_DeveInvalidarOCadastro**

Passa para o controlador de cadastro de veículo dados de um veículo, porém com uma placa que já pertence a um carro no banco, depois chama o método de validar o cadastro, e verifica se cadastro foi invalidado (campo que indica se o cadastro está válido é falso).

##### **testCadastrarVeiculo\_ValidarPlaca\_PlacaNaoCadastrada\_DeveValidarOCadastro**

Passa para o controlador de cadastro de veículo dados de um veículo, com uma placa que não pertence a um carro no banco, depois chama o método de validar o cadastro, e verifica se cadastro foi validado (campo que indica se o cadastro está válido é verdadeiro).

#### **4.3 - Testes para reserva de carro**

##### **4.3.1 - Testes Unitários**

##### **Validador: testValidarCPF\_CPFNulo\_DeveRetornarFalse**

Passa para o validador de reserva uma entrada com um CPF nulo e espera que a entrada seja rejeitada.

**Validador: testValidarCPF\_CPFNaolInserido\_DeveRetornarFalse**

Passa para o validador de reserva uma entrada com um CPF em branco e espera que a entrada seja rejeitada.

**Validador: testValidarModelo\_ModeloNulo\_DeveRetornarFalse**

Passa para o validador de reserva uma entrada com um modelo nulo e espera que a entrada seja rejeitada.

**Validador: testValidarModelo\_ModeloNaolInserido\_DeveRetornarFalse**

Passa para o validador de reserva uma entrada com um modelo em branco e espera que a entrada seja rejeitada.

**Validador: testValidarMarca\_MarcaNulo\_DeveRetornarFalse**

Passa para o validador de reserva uma entrada com marca nula e espera que a entrada seja rejeitada.

**Validador: testValidarMarca\_MarcaNaolInserida\_DeveRetornarFalse**

Passa para o validador de reserva uma entrada com marca em branco e espera que a entrada seja rejeitada.

**Validador: testValidarDataFim\_DataFimNula\_DeveRetornarFalse**

Passa para o validador de reserva uma entrada com a data fim nula e espera que a entrada seja rejeitada.

**Validador: testValidarDataFim\_DataInicioNula\_DeveRetornarFalse**

Passa para o validador de reserva uma entrada com a data início nula e espera que a entrada seja rejeitada.



**Validador: testValidarDataFim\_DataFimAntesDaDataInicio\_DeveRetornarFalse**

Passa para o validador de reserva uma entrada com a data fim anterior à data início e espera que a entrada seja rejeitada.

**Validador: testValidarDataFim\_DataFimDepoisDaDataInicio\_DeveRetornarTrue**

Passa para o validador de reserva uma entrada com a data fim posterior à data início e espera que a entrada seja aceita.

#### **4.3.2 - Testes Funcionais**

**Validador:**

**testValidadorDadosReservaVeiculo\_ValidarMarcaEModelo\_DadosCorretos\_DeveRetornarTrue**

Passa para o validador de reserva uma entrada com marca e modelo que correspondem a um carro no banco e espera que a entrada seja aceita.

**Validador:**

**testValidadorDadosReservaVeiculo\_ValidarMarcaEModelo\_DadosIncorretos\_DeveRetornarFalse**

Passa para o validador de reserva uma entrada com marca e modelo que não correspondem a um carro no banco e espera que a entrada seja rejeitada.

**Controlador:**

**testControladorReservaVeiculoPorModelo\_PopularVeiculo\_MaisDeUmVeiculoDa  
MesmaMarcaEModelo\_DeveEscolherQualquerUm**

Passa para o controlador de reserva uma entrada com marca e modelo que correspondem a mais de um carro no banco e espera que um deles seja selecionado.

**Controlador:**

**testControladorReservaVeiculoPorModelo\_Cadastrar\_VeiculoDeveEstarDisponiv  
elDepoisDeEfetuarAReserva**

Usa o sistema para cadastrar um veículo depois tenta reservar o mesmo veículo. Espera que o veículo já esteja disponível para ser reservado.

**Controlador:**

**testControladorReservaVeiculoPorModelo\_NenhumCarroDisponivelParaMarcaEM  
odelo\_NaoDeveFazerReserva**

Passa para o controlador de reserva uma entrada com marca e modelo que correspondem a um ou mais carros no banco, mas sem nenhum veículo disponível. Espera que a reserva não seja realizada.

**Controlador: testValidarCPF\_CPFNaolInserido\_DeveInvalidarReserva**

Passa para o controlador de reserva uma entrada com CPF nulo e chama o método de validar reserva, depois verifica se o campo que indica se a reserva está válida é falso.

**Controlador: testValidarModelo\_ModeloNaolInserido\_DeveInvalidarReserva**

Passa para o controlador de reserva uma entrada com modelo nulo, chama o método de validar reserva e espera que o campo que indica se a resrva está válida seja falso.

**Controlador:**

**testControladorReservaVeiculo\_VerificarClienteCadastrado\_ClienteNaoCadastrado\_DeveInvalidarAReserva**

Passa para o controlador de reserva uma entrada com um CPF de um cliente que não existe no banco, chama o método de validar reserva e espera que o campo que indica se a reserva está válida seja falso.

**Controlador:**

**testControladorReservaVeiculo\_VerificarClienteCadastrado\_ClienteCadastrado\_DeveValidarAReserva**

Passa para o controlador de reserva uma entrada com um CPF de um cliente que existe no banco, chama o método de validar reserva e espera que o campo que indica se a reserva está válida seja verdadeiro.

**Controlador:**

**testControladorReservaVeiculo\_VerificarClienteListaNegra\_ClienteEstaNaListaNegra\_DeveInvalidarAReserva**

Passa para o controlador de reserva uma entrada com um CPF de um cliente que está na lista negra, chama o método de validar reserva e espera que o campo que indica se a reserva está válida seja falso.

**Controlador:**

**testControladorReservaVeiculo\_VerificarClienteListaNegra\_ClienteNaoEstaNaListaNegra\_DeveValidarAReserva**

Passa para o controlador de reserva uma entrada com um CPF de um cliente que não consta na lista negra, chama o método de validar reserva e espera que o campo que indica se a reserva está válida seja verdadeira.

#### **4.4 Testes para a consulta de clientes:**

#### **testControladorConsultaCliente\_FiltrarClientes\_CPFNulo**

Passa para o filtro de clientes uma entrada com CPF nulo e espera que ele retorne todos os clientes.

#### **testControladorConsultaCliente\_FiltrarClientes\_CPFNaoNulo**

Passa para o filtro de clientes uma entrada com CPF não nulo e espera que ele retorne apenas o cliente em questão.

#### **testControladorConsultaCliente\_FormatarInformacoesCliente\_DeveFormatarCPF ENome**

Chama o método de formatação de informações do cliente no controlador de consulta de cliente e espera que o texto retornado esteja formatado adequadamente.

### **4.5 Testes para a consulta de reservas;**

#### **testControladorConsultaReserva\_FiltrarReservas\_CPFNaoNulo\_DeveTrazerTodasAsReservasDoCliente**

Passa para o filtro de reservas uma entrada com CPF não nulo e espera que ele retorne todas as reservas do cliente dono do CPF digitado. Não deve encontrar reservas de outro cliente.

#### **testControladorConsultaReserva\_FormatarReserva\_DeveFormatarInformacoesDaReserva**

Chama o método de formatação de informações da reserva no controlador de consulta de reserva e espera que o texto retornado esteja formatado adequadamente.

### **4.6 Testes para as informações do cliente**

### **testControladorInformacoesCliente\_AgregarListaNegra\_ClienteDeveAparecerNaListaNegra**

Ao verificar as informações de um cliente que não está na lista negra, chama o método que adiciona o cliente na lista negra, depois verifica se o cliente aparece na lista negra (deve encontrar o cliente).

### **testControladorInformacoesCliente\_RemoverListaNegra\_ClienteNaoDeveAparecerNaListaNegra**

Ao verificar as informações de um cliente que já está na lista negra, chama o método que remove o cliente da lista negra, depois verifica se o cliente aparece na lista negra (não deve encontrar o cliente).

## **4.7 Testes para as informacoes da reserva**

### **testControladorInformacoesReserva\_Cancelar\_ReservaPodeSerCancelada\_DeveRemoverAReserva**

Verifica as informações de uma reserva que cuja data de início ainda não passou, chama o método de cancelamento de reserva, depois verifica se a reserva consta no sistema (não deve encontrar a reserva).

### **testControladorInformacoesReserva\_Cancelar\_ReservaNaoPodeSerCancelada\_NaoDeveRemoverAReserva**

Verifica as informações de uma reserva que cuja data de início já passou, chama o método de cancelamento de reserva, depois verifica se a reserva consta no sistema (deve encontrar a reserva).

### **ControladorInformacoesReserva\_ReservaPodeSerCancelada\_DataInicioNoFuturo\_ReservaPodeSerCancelada**

Verifica as informações de uma reserva que cuja data de início ainda não passou, chama o método que verifica se a reserva pode ser cancelada. Espera que o método retorne verdadeiro.

#### **ControladorInformacoesReserva\_ReservaPodeSerCancelada\_DataInicioNoPassado\_ReservaNaoPodeSerCancelada**

Verifica as informações de uma reserva que cuja data de início já passou, chama o método que verifica se a reserva pode ser cancelada. Espera que o método retorne falso.

#### **ControladorInformacoesReserva\_ReservaPodeSerCancelada\_DataInicioNoPresente\_ReservaPodeSerCancelada**

Verifica as informações de uma reserva que cuja data de início é a data atual, chama o método que verifica se a reserva pode ser cancelada. Espera que o método retorne verdadeiro.

### **4.8 Testes para as informacoes do veículo**

#### **testControladorInformacoesVeiculo\_EnviarManutencao\_DeveEnviarOVeiculoParaManutencao**

Chama o método de enviar veículo para a manutenção e espera que o veículo fique indisponível.

#### **testControladorInformacoesVeiculo\_RemoverManutencao\_DeveRemoverOVeiculoDaManutencao**

Chama o método de retornar veículo da manutenção e espera que o veículo fique disponível.

### **4.9 Testes para o pagamento**

### **testControladorPagamento\_VenderCarro\_FormaPagamentoCreditoEmXVezes\_DeveSetarFormaDePagamentoParaCredito**

Chama o método do pagamento com o cartão de crédito e verifica que é possível pagar parcelado.

### **testControladorPagamento\_VenderCarro\_FormaPagamentoDebito\_DeveSetarFormaDePagamentoParaDebito**

Chama o método do pagamento com o cartão de débito e verifica que a forma de pagamento foi setada para débito

### **testControladorPagamento\_VenderCarro\_FormaPagamentoDinheiro\_DeveSetarFormaDePagamentoParaDinheiro**

Chama o método do pagamento com o cartão de dinheiro e verifica que a forma de pagamento foi setada para dinheiro.

### **testControladorPagamento\_VenderCarro\_VeiculoIndisponivel\_DeveSetarVendidoParaTrue**

Em um veículo que está indisponível, chama o método de vender veículo. Espera que o status de vendido do veículo passe para verdadeiro.

### **testControladorPagamento\_PagarAluguel\_DeveSetarPagoParaTrueEFormaPagamentoParaDinheiro**

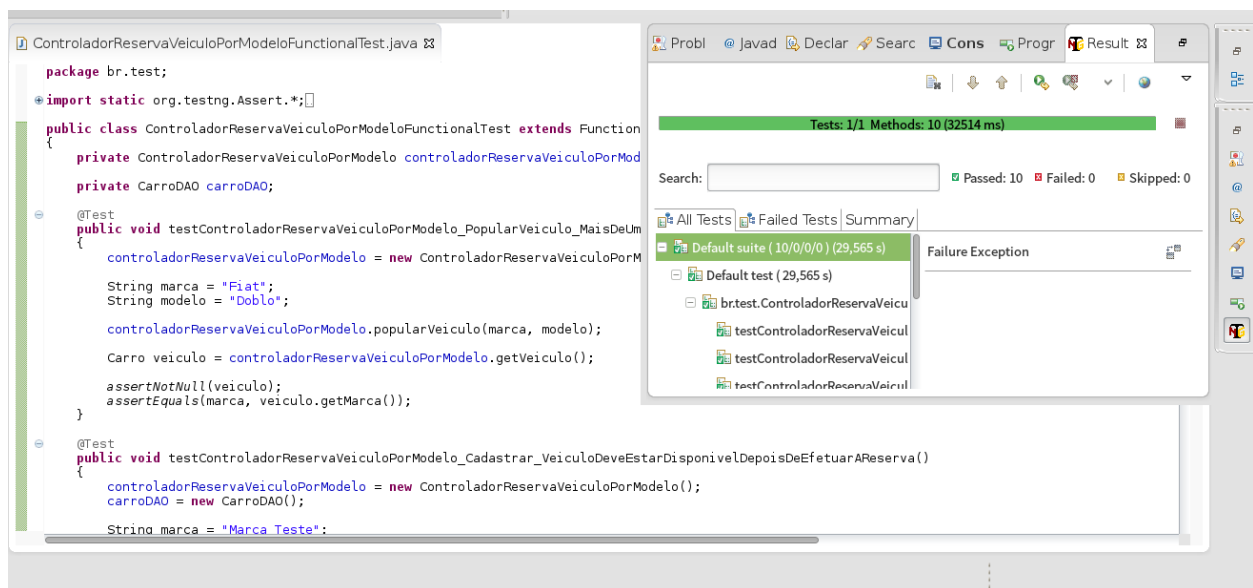
Chama o método para o pagamento de aluguel e verifica que foi setado a forma de pagamento para dinheiro

### **testControladorPagamento\_VenderCarro\_VeiculoDisponivel\_DeveSetarVendidoParaTrueEDisponivelParaFalse**

Em um veículo que está disponível, chama o método de vender veículo. Espera que o status de vendido do veículo passe para verdadeiro e que o de disponível passe para falso.

#### 4.10 Imagens dos testes

Abaixo encontram-se algumas imagens de algumas classes de teste, as classes podem ser encontradas nesse mesmo zip.





```
ControladorCadastroClienteFuncionalTest.java
public class ControladorCadastroClienteFuncionalTest extends FunctionalTest
{
    private ControladorCadastroCliente controladorCadastroCliente;
    private PessoaDAO pessoaDAO;

    @Test
    public void testCadastrarCliente_CadastrarComSucesso_ClienteDeveEstarNoBanco()
    {
        controladorCadastroCliente = new ControladorCadastroCliente();
        pessoaDAO = new PessoaDAO();

        controladorCadastroCliente.cadastrar("Ana Mel da Cohab", "12345678", "11111111", "11111", "A", "11111111", "emaildaanamel@email.com");

        Pessoa pessoa = pessoaDAO.recuperarPessoaPorCPF("12345678");

        assertEquals(pessoa.getNome(), "Ana Mel da Cohab");
        assertEquals(pessoa.getCpf(), "12345678");
        assertEquals(pessoa.getRg(), "11111111");
        assertEquals(pessoa.getCarteira(), "11111");
        assertEquals(pessoa.getCategoriaCarteira().toString(), "A");
        assertEquals(pessoa.getTelefone(), "11111111");
        assertEquals(pessoa.getEmail(), "emaildaanamel@email.com");

        limparPessoaAdicionadaPorCPF("12345678");
    }

    @Test
    public void testVerificarCPF_CPFNaoExisteNoBanco_DeveRetornarTrue()
    {
        controladorCadastroCliente = new ControladorCadastroCliente();

        assertTrue(controladorCadastroCliente.verificarCPF("123123123"));
    }
}
```

```
ValidadorDadosCadastroVeiculoUnitTest.java
ValidadorDadosCadastroClienteUnitTest.java

public class ValidadorDadosCadastroClienteUnitTest
{
    private ValidadorDadosCadastroCliente validadorDadosCadastroCliente = new ValidadorDadosCadastroCliente();

    @Test
    public void testValidarNome_NomeComMaisDe3Letras_DeveRetornarTrue()
    {
        String nome = "Carlos";

        Assert.assertTrue(validadorDadosCadastroCliente.validarNome(nome));
    }

    @Test
    public void testValidarNome_NomeComMenosDe3Letras_DeveRetornarFalse()
    {
        String nome = "Ca";

        Assert.assertFalse(validadorDadosCadastroCliente.validarNome(nome));
    }

    @Test
    public void testValidarNome_NomeNaoPodeTerNumeros_DeveRetornarFalse()
    {
        String nome = "Carlos123";

        Assert.assertFalse(validadorDadosCadastroCliente.validarNome(nome));
    }

    @Test
    public void testValidarNome_NomeNulo_DeveRetornarFalse()
    {
        String nome = null;
    }
}
```

Tests: 1/1 Methods: 20 (4602 ms)

Search:  Passed: 20 Failed: 0 Skipped: 0

All TestsFailed TestsSummary

testValidarRG\_RGNulo\_DeveRetornarTrue

testValidarRG\_RGSemLetras\_DeveRetornarTrue

testValidarTelefone\_TelefoneComMaisDe15Digitos\_DeveRetornarFalse

testValidarTelefone\_TelefoneComMenosDe15Digitos\_DeveRetornarTrue

testValidarTelefone\_TelefoneNulo\_DeveRetornarFalse

testValidarTelefone\_TelefoneSemArea\_DeveRetornarTrue

Failure Exception

## 5. Considerações Finais

Com o projeto Voce-Aluga tivemos a oportunidade de por em prática vários conhecimentos de desenvolvimento de sistemas, assim como várias boas práticas. Experienciar o desenvolvimento de um sistema completo usando a metodologia RUP e usar do conhecimento dos próprios integrantes da equipe para mesclar esse método com práticas de metodologias Ágeis.

Tivemos a oportunidade de por em prática estratégias de desenvolvimento como o TDD e usufruir das facilidades do JPA combinado com o Hibernate.

Para finalizar, durante todo o desenvolvimento fizemos forte uso de estratégias de versionamento e do servidor Git combinado com o repositório Github. Todo o projeto pode ser encontrado no link <https://github.com/csmartins/VoceAluga> . Basta clonar o repositório para ter acesso a todo o conteúdo do projeto.

