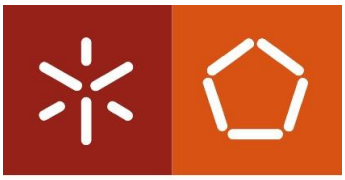

Universidade do Minho

Programação Orientada a Objetos

UMCarroJá

Grupo Nº 48

João Teixeira (A85504)

Emanuel Rodrigues (A84776)

José Ferreira (A83683)

26 de Maio de 2019

Conteúdo

Capítulo 1

Introdução

O objetivo deste projeto é construir um sistema de aluguer de carros, inspirado no serviço de aluguer de casas *Airbnb*, onde um cliente pode alugar um carro, onde ele mesmo e o condutor, para fazer a deslocação que pretende, ou disponibilizar as suas viaturas para alugar. Para a realização deste projeto vamos aplicar conhecimentos adquiridos nas aulas da UC de Programação Orientada a Objetos. Ao longo deste relatório vamos descrever a nossa abordagem a este problema.

Capítulo 2

Arquitetura e Solução do Projeto

Para a construção deste projeto decidimos implementar o padrão de arquitetura de software *MVC*. Assim, é consideravelmente mais fácil implementar qualquer alteração na interface de utilizador ou no *Model*.

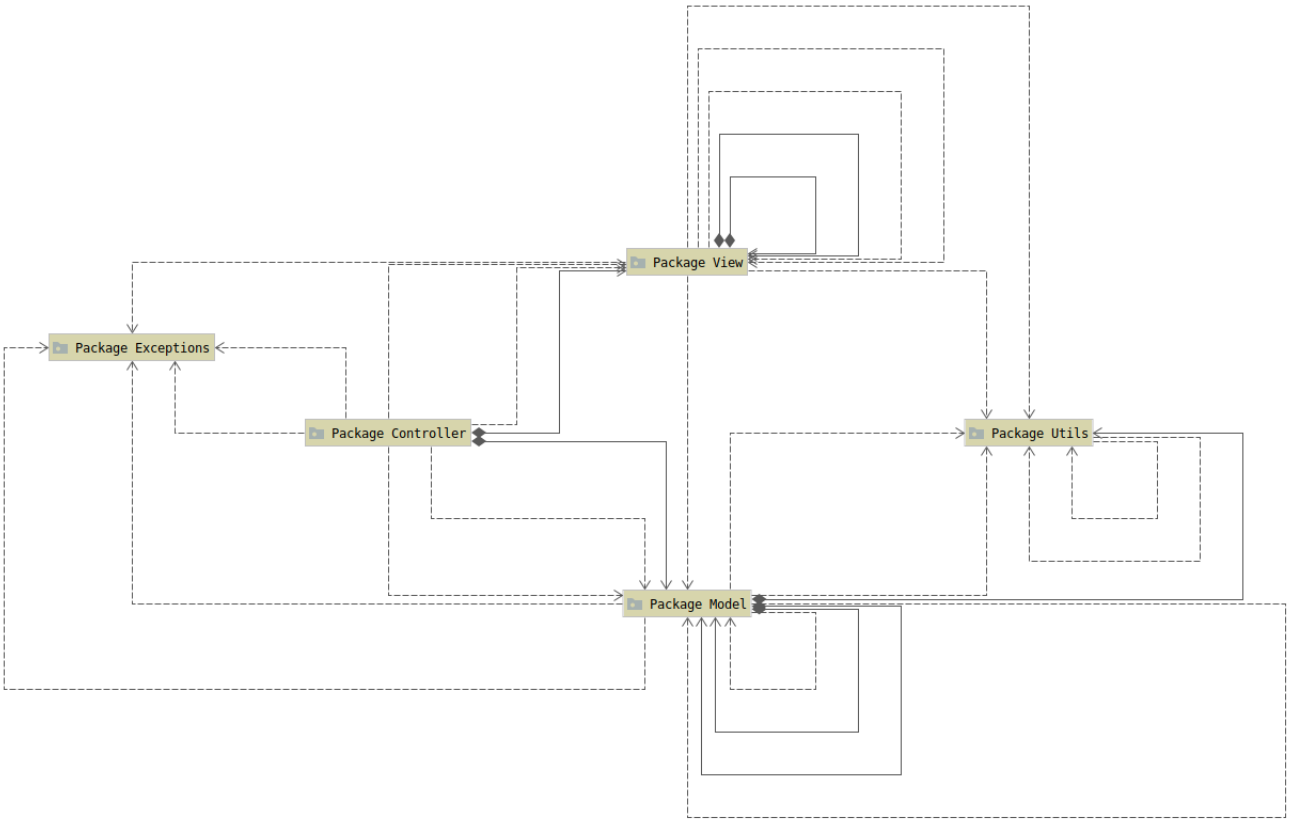

Figura 2.1: Arquitetura do Projeto

Capítulo 3

Classes

3.1 Modelo

No modelo todos os pedidos entram pela classe UmCarroJá como visível na hierarquia de classes desta, na figura 3.1, salvo exceções, como alguns gets sobre classes deste. Daqui são chamados todos os métodos das outras classes necessários a realizar os pedidos.

3.1.1 UMCarroJá

Esta e a classe onde esta contida toda a informação sobre utilizadores, carros e alugueres. É também a grande ponte de comunicação com o exterior do modelo, permitindo assim que não haja interação direta do exterior com as classes.

3.1.2 User

Esta e a classe com a informação contida por qualquer user do sistema, e métodos comuns tanto aos clientes como aos owners.

3.1.3 Client

Esta classe é referente ao user que pode criar alugueres, contendo esta um Ponto, correspondente à posição em que se encontra e alugueres que ainda não foram avaliados.

3.1.4 Owner

Esta classe é relativa ao utilizador que tem os seus carros para aluguer, e este tem informação sobre os carros que possui e também os alugueres que ainda não avaliou.

3.1.5 Users

Esta classe contem a informação de todos os utilizadores do sistema.

3.1.6 Car

Esta classe representa uma viatura, onde tem todas as suas informações, desde autonomia, quem é o seu proprietário, marca e matricula.

3.1.7 Cars

Esta classe trata de guardar todos os carros existentes no sistema, bem como procurar carros conforme as condições dadas para efetuar um aluguer.

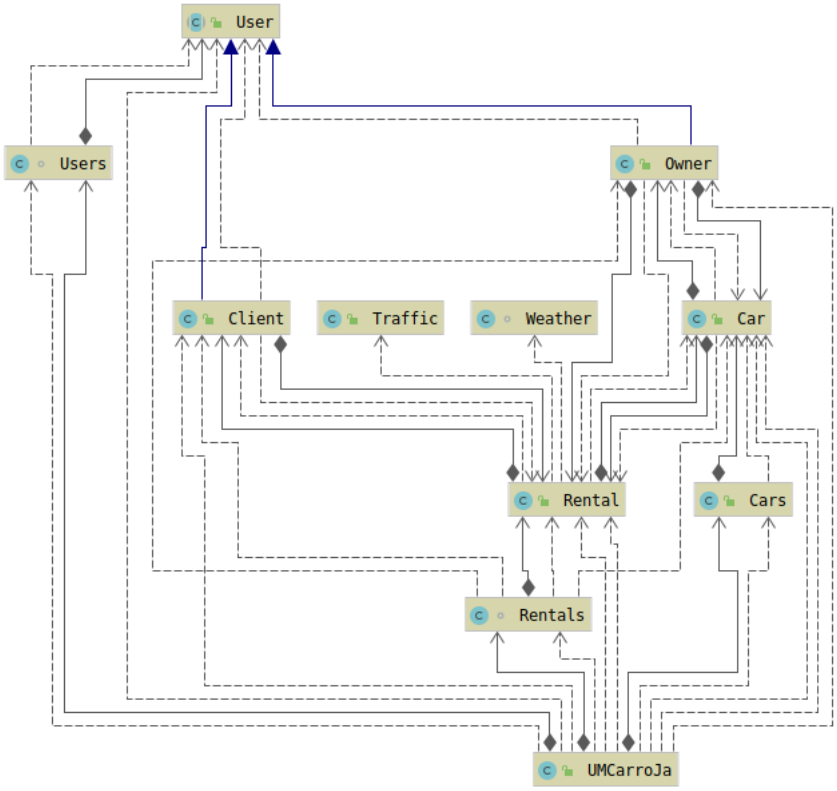

Figura 3.1: Hierarquia de Classes do **UmCarroJá**

3.1.8 Rental

Esta classe contem toda a informação de um Aluguer assim como métodos para tratar um aluguer.

3.1.9 Rentals

Esta classe guarda a informação relativa a todos os alugueres efetuados no sistema.

3.1.10 Parser

Esta classe tem como objetivo a leitura do ficheiro de logs e transformação em informação utilizável pelo programa.

3.2 View

3.2.1 Menu

Esta classe representa os vários Menus e as relações entre eles. Para permitir conhecer o caminho percorrido até ao menu que se está a observar esta classe contém uma stack com os menus percorridos.

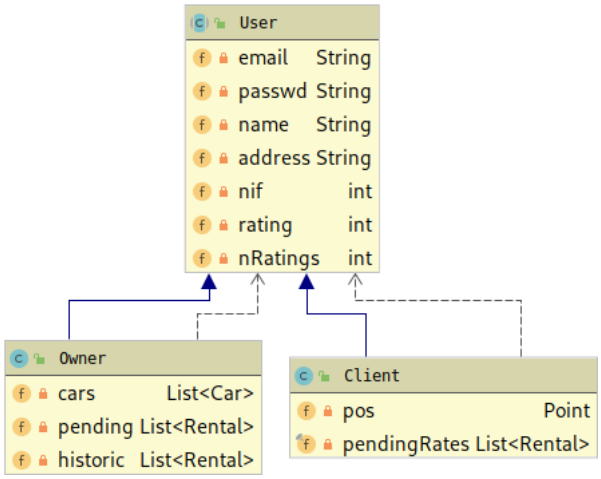

Figura 3.2: Hierarquia de Classes do **User**

3.2.2 Table

Esta classe representa uma Tabela com um generic type parameter. Para tal apenas precisa de uma lista de Labels para as colunas, outra para as linhas e os dados sobre a forma de uma Lista de Listas. O resultado final é uma tabela visualmente apelativa em que cada coluna tem o tamanho mínimo possível.

3.2.3 ViewModel

Esta Package contem um conjunto de classes que permitem a transferência de argumentos entre a View e o Controller de forma mais legível. Por exemplo, a classe RegisterCar contém todos os parâmetros necessários para registar um novo carro e uma instância desta é passada da View para o Controller a fim de o adicionar aos carros no model.

3.3 Controller

Cria a ponte entre o View e o Model. Assim, o Controller é o único que conhece a view e o Model, sendo que tanto a View como o Model apenas conhecem o Controller.

3.4 Utils

3.4.1 Point

Esta classe contém as coordenadas de um ponto num plano, assim como métodos sobre estes. Por exemplo, um método para calcular a distância entre dois pontos.

3.4.2 StringBetter

Esta classe contém métodos para converter uma string para quando for impressa num ecrã tenha várias propriedades. Tais como terem cor específica e estarem sublinhadas.

3.5 Exceptions

Esta Package contém todas as classes de exceções utilizadas ao longo do projeto.

Capítulo 4

Introdução de novos tipos de Viaturas

Neste momento, e à luz do ficheiro de logs fornecido, todos os carros são tratados de igual maneira, pelo que a diferenciação entre os diversos tipos de veículos e feita através de um *enum*. Para facilitar a criação de novos tipos de veículos, a nossa classe *Car* devia ser tornada numa interface, ou uma classe abstrata, de forma a que inserir novos veículos fosse apenas a criação de uma nova classe que implementa ou extende a classe *Car*

Capítulo 5

Manual de Utilização

No menu inicial apresentado é possível escolher se se quer fazer login ou criar novo utilizador. Visto que existem dois tipos distintos de utilizadores(Clientes e Proprietários), existem dois menus diferentes para cada tipo de utilizador. Essa seleção é feita automaticamente a quando do login. Dentro do menu de Proprietários é possível ver o histórico de alugueres, aprovar alugueres pendentes, adicionar carros e realizar várias operações sobre carros (reabastecer o carro, mudar o preço, mudar disponibilidade e calcular o rendimento de um carro). Dentro do menu de Clientes é possível ver o histórico de alugueres, ver classificações pendentes, alugar um carro e ver os 10 melhores clientes. Para navegar para trás no menu basta escrever o carácter *b*.

Capítulo 6

Conclusão

Para concluir, conseguimos cumprir todos os requisitos propostos criando no processo um sistema semelhante ao *Airbnb* para o aluguer de carros que segue um modelo *MVC* (Model View Controller) Como trabalho futuro, gostaríamos de melhorar a forma como a classe *Car* está implementada a fim de facilitar a adição de novos tipos de veículos.