FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

sexta-feira, 23 de novembro de 2018

Hotel AEDA

Relatório Algoritmos e Estruturas de Dados
2018/19

2MIEIC07_03:

Gonçalo Marantes up201706917@fe.up.pt
João Santos up201707427@fe.up.pt
Luís Gonçalves up201706760@fe.up.pt

ÍNDICE

Descrição

Este programa está projetado de forma a organizar as ações mais básicas de um hotel, tais como gestão de clientes, reservas e empregados e até mesmo da própria infraestrutura do hotel, como quartos, andares e localização. Para além destas funcionalidades, o Hotel também dispõe aos seus clientes informações sobre restaurantes e serviço de excursões a pontos turísticos.

Assim, tendo ao seu dispor informações como o número de andares e a localização é possível para o gerente utilizar o programa na gestão de espaços do seu hotel. A partir desse ponto é possível criar uma ficha de clientes, fazer check-in e check-out dos mesmos, fazer e cancelar reservas, gerir quartos e sala de encontros e empregados. Pode, também, consultar as informações sobre cada cliente, como o nome e data de aniversário, reserva, como o quarto reservado e preço, quartos e sala de reuniões, como o preço preço associado e a capacidade, e empregado, tais como número de identificação e nome.

Solução Implementada

De forma a estruturar o código da melhor forma possível, foram criados 10 módulos, cada um com o seu propósito e no caso de se tratar de uma classe, o seu objeto. Sendo que um dos módulos é dedicado às interações com o utilizador.

- **Hotel** – Todas as estruturas de dados que são usadas pelo programa são geridas neste elemento;
- **Room** – Uma classe abstrata que possui como classes derivadas Bedroom e MeetingRoom. Esta implementação teve origem no facto de tal como os nomes indicam, existirem elementos em comum entre um quarto e uma sala de encontros;
- **Client** – Possui a informação sobre cada cliente;
- **Reservation** – Condensa a informação sobre cada reservas;
- **Employee** – Agrega a informação sobre cada empregado;
- **Date** – Classe auxiliar com informação relativa à data;
- **Event** – Contém informação sobre eventos a decorrer;
- **Restaurant** – Contém informações relativamente a restaurantes;
- **Van** – Possui informações de cada carrinha relativa às excursões.

Estruturas de Dados Implementadas

Para guardar e organizar toda esta informação implementamos as seguintes estruturas de dados:

`vector<Client*> clientsCheckedIn`

Vetor que contém apontadores para os atuais clientes do hotel.

`hashTabClientRecords clientRecords`

Hash Table baseada na estrutura `unordered_set` que contém apontadores para todos os clientes do hotel, tanto atuais como antigos.

`vector<Reservation> reservations`

Vetor que contém todo o histórico de reservas feitas por todos clientes.

`vector<Room*> rooms`

Vetor que contém apontadores para todos os quartos, tanto do tipo Bedroom como MeetingRoom, do hotel.

`vector<Employee> employees`

Vetor que contém todos os empregados do hotel.

`priority_queue<Van> vans`

Fila de prioridade contendo as carrinhas, as mesmas estão organizadas pelo número de vagas, isto é, está no início da fila a carrinha com o menor número de vagas.

`priority_queue<Event> events`

Fila de prioridade contendo todos os eventos, os mesmos estão organizados por data, isto é, está no início da fila o evento com a data mais próxima da atual.

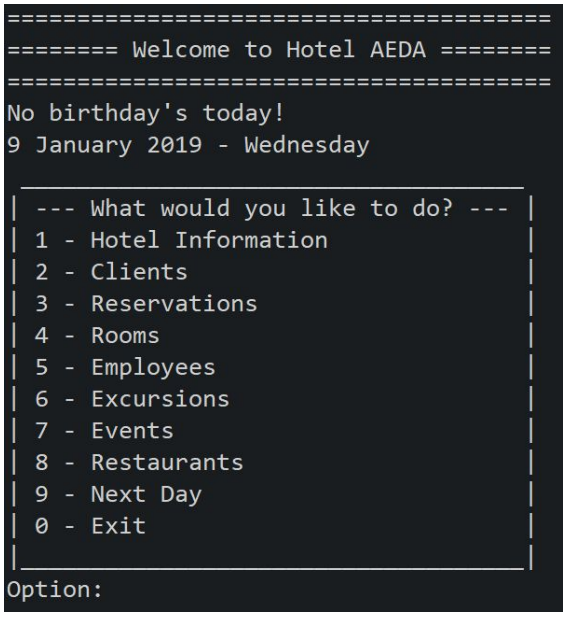
`BST<Restaurant> restaurants`

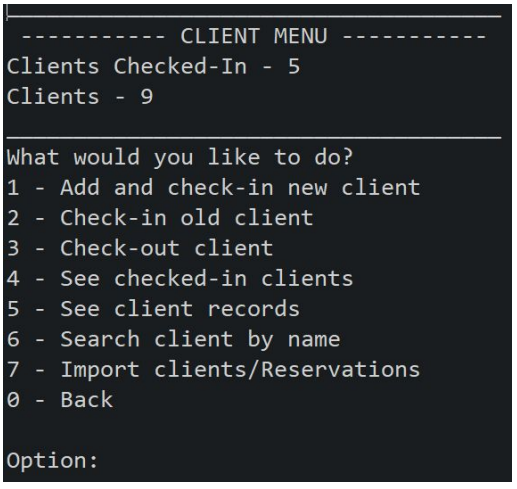
Árvore binária de pesquisa contendo restaurantes para apoio aos clientes do hotel. Os restaurantes estão organizados alfabeticamente pelo seu tipo de cozinha e em caso de empate pela sua distância ao hotel.

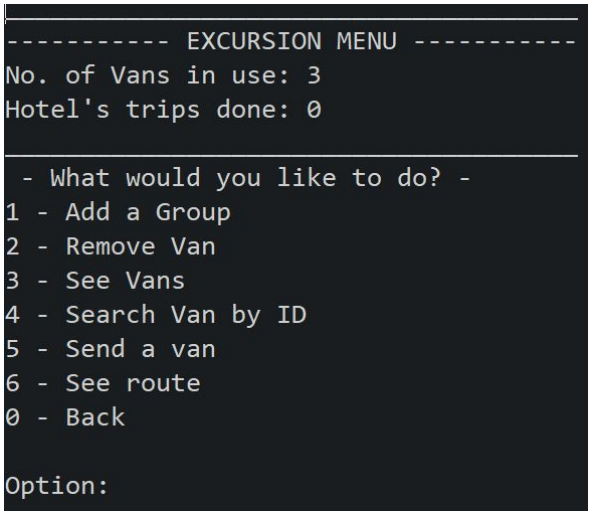
Descrição dos Módulos

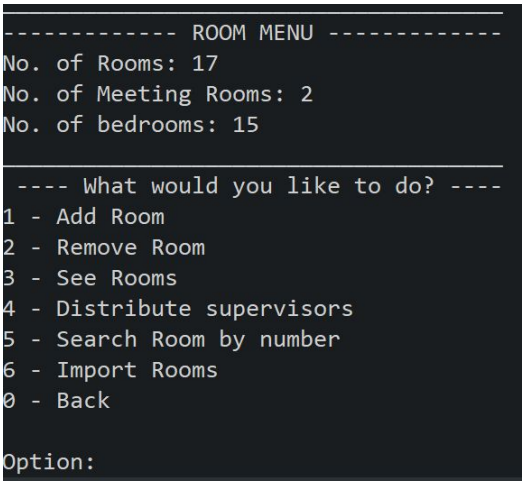
Menu:

O Menu é uma classe que serve de interface entre o utilizador e programa. Esta classe retira de um ficheiro de texto a informação do hotel e cria as opções lá especificadas, de forma mutável (baseado nos conteúdos do ficheiro de texto) e hierarquizada (a opção 1.1 surge após selecionar a opção 1, etc).



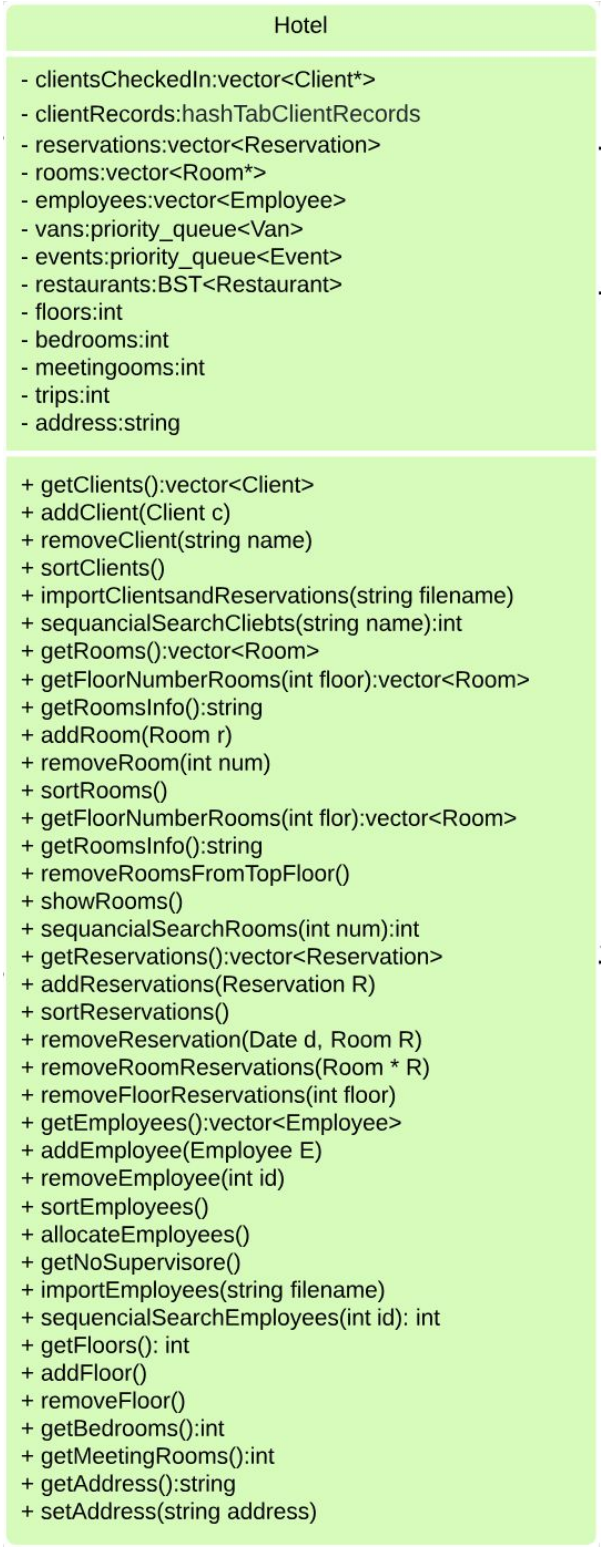






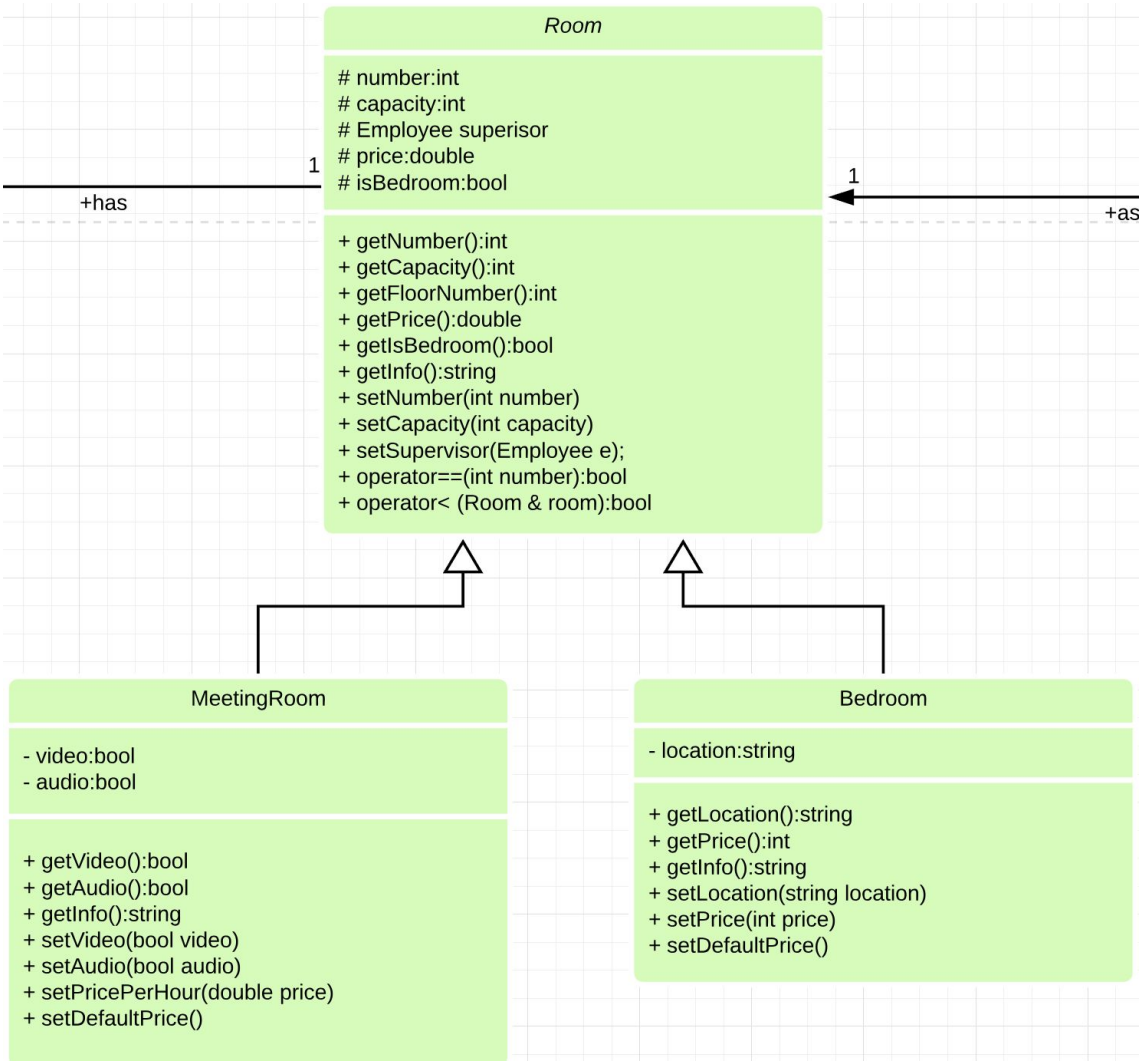
Hotel:

É nesta classe que tudo se inicia. Todas as estruturas essenciais são criadas e vão ser utilizadas a partir desta classe. Esta classe vai gerir tudo, a adição, eliminação e alteração de clientes, empregados, quartos e salas de encontros e reservas. Isto procede, então, a classe Menu.



Room:

Esta classe é uma classe abstrata que possui como classes derivadas as classes: Bedroom (quarto) e MeetingRoom (sala de encontro). Por serem bastante semelhantes foi a decisão do grupo torná-las subclasses da superclasse Room. Todas as Rooms possuem em seus parâmetros o número, capacidade, supervisor e preço. A partir deste momento passamos para as subclasses nas quais a Bedroom possui mais um parâmetro, a sua localização, e a MeetingRoom possui mais dois parâmetros, a existência ou não de vídeo e áudio na sala.



Client:

A classe Client representa uma cliente, possui membros de dados como o nome e a lista de reservas feita por esse cliente.

Client
- id:int - name:string - reservations:vector<Reservation*> - nextId:static int
+ getName():string + setName(string name) + getId():int + setId(int id) + getBirthday() + getInfo():string + getInfo2():string + getReservations():string + addReservation(Reservation reservations) + removeReservation(Date d, Room *r)

Reservation:

Possui membros de dados como a data, o quarto/sala de encontros, duração da estadia e o preço final.

Reservation
- date:Date - room:Room* - duration:int - price:double
+ getDate():Date + getRoom():Room* + getDuration():int + getPrice():double + getInfo():string + setDate(Date date) + setRoom(Room *room) + setPrice(double price) + setDuration(int duration) + operator <(Reservation & r):bool + operator =(Reservation & r):Reservation

Employee
- name:string - id:int - isSupervisor:bool
+ getId():int + getName():string + getIsSupervisor():bool + getInfo():string + setId(int id) + setName(string name) + setIsSupervisor(bool supervisor); + operator == (int id):bool + operator == (string name):bool + operator < (Employee & e):bool

Employee:

Possui membros de dados como a id, o nome e a sua classificação de supervisor.

Date:

Por questões de facilidade de implementação, foi criada esta classe de forma a organizar melhor a estrutura da data, possui como membros de dados o dia, o mês, o ano, a época do ano e o dia da semana.

Date
- day:int - month:int - year:int - season:string - weekday:string
+ getDay():int + getMonth():int + getYear():int + getSeason():string + getWeekday():string + getInfo():string + setDay(int day) + setMonth(int month) + setYear(int year) + setDate(string &date) + setSeason(string season) + setWeekday(string weekday) + showDate() + showExtendedDate() + operator ++():Date + operator ++(int):Date + operator --():Date + operator --(int):Date + operator ==(Date &date):bool + operator =(Date date):Date*

Van:

A classe van representa uma carrinha, cada carrinha possui os membros-dados id (um número de identificação), capacity (capacidade total da carrinha), vacancies (número de lugares vagos na carrinha) e o vetor onBoard (um vetor de apontadores para os clientes do hotel, são os clientes que estão inscritos para fazerem a viagem naquela carrinha).

Van
- id:int - capacity:int - vacancy:int - onBoard:vector<Client*> - nextId:static int
+ getId():int + setId(int id) + getCapacity():int + getVacancies():int + addGroup(vector<Client*>& group) + getInfo():string

Event:

Devido à política do Hotel AEDA, a gerência envia e-mails a congratular o dia de aniversário de todos os seus clientes. Os conteúdos desses e-mails envolvem convites para eventos futuros do Hotel, sendo assim foi necessário criar o objeto Event. Este objeto tem a seguinte estrutura:

Event
- name:string - date:Date - location:string - description:string
+ getDate():Date + getName():string + getLocation():string + getDescription():string + getBirthday() + getInfo():string

Restaurant:

A classe restaurant representa um restaurante e possui os membros-dado name (o nome do restaurante), type (o tipo de cozinha do restaurante, ex. italiana) e distance (distância entre o restaurante e o hotel em quilómetros).

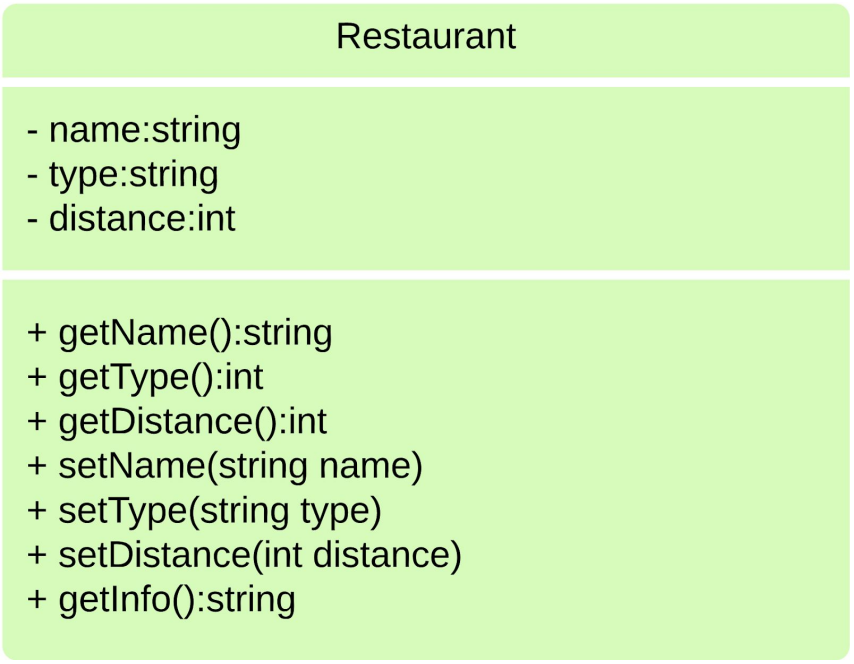
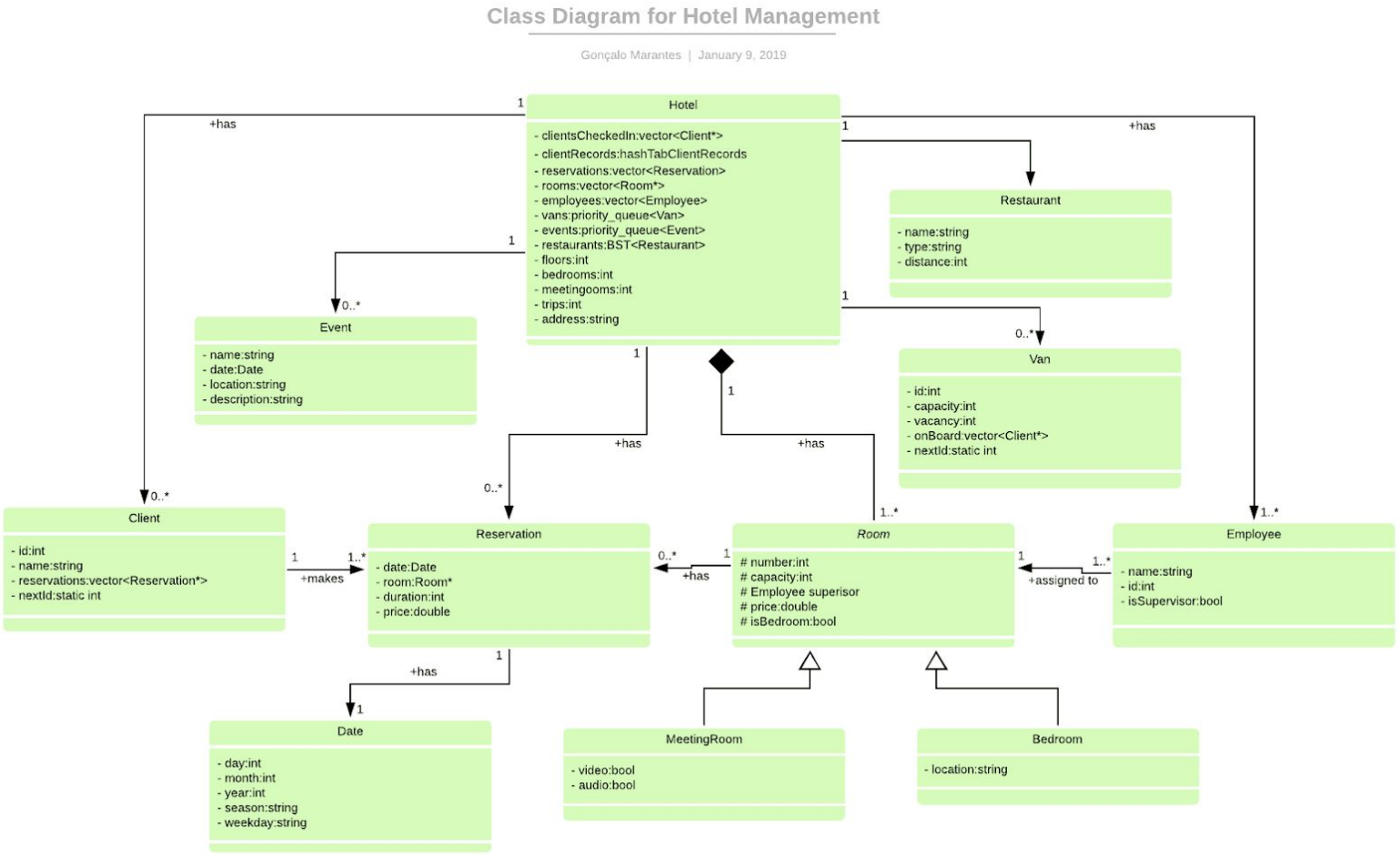


Diagrama UML Simplificado

Uma vez que o Diagrama de Classes Original é demasiado extenso para colocar uma imagem do mesmo neste relatório, fornecemos uma versão simplificada. Esta versão contém apenas os membros-dado.



Principais dificuldades

Não tivemos dificuldades específicas que tenham sido pronunciadas, exceto que o programa não corria no computador de um dos nossos membros, tivemos problemas de compatibilidades entre diferentes IDE, tendo ficado ele encarregado pelas partes em que não eram necessárias testar o programa.

Durante a implementação do membro função `getCurrentDate()` da classe `Date`, utilizamos a função `localtime_s()` da biblioteca <time.h>, no entanto a mesma não era reconhecida pelo compilador MinGW, tendo então sido mudada para `localtime()` da biblioteca de C <ctime>.

Distribuição do trabalho

O trabalho foi destruído o mais equitativamente possível, sendo que o módulo principal (Hotel) foi realizado por todos consoante as implementações dos módulos auxiliares, como é possível ver:

Gonçalo Marantes – Módulos Client, Date, Event, Room, Reservation

Luís Gonçalves – Módulos Employee, Restaurant, Menu

João Santos – Módulos Van, realização do relatório e documentação do código

Conclusão

Em suma, apesar de alguns problemas de compatibilidade, foi bastante interessante aplicarmos os conceitos teóricos adquiridos nesta disciplina numa situação real, de forma a ganharmos uma melhor perceção dos casos de utilização de diferentes estruturas de dados. Mostrando-nos de uma forma mais concreta aquilo que será exigido de nós.