T7 – Empresa de Táxis

Algoritmos e Estrutura de Dados

Nuno Manuel Ferreira Corte-Real – 201405158 – up201405158@fe.up.pt

20 de Novembro de 2016

Turma – 2MIEIC03 – Grupo - G

Faculdade de Engenharia da Universidade do Porto

Rua Roberto Frias, sn, 42''-465 Porto, Portugal

Índice

Empresa de Táxis

Este trabalho tem como tema uma empresa de táxis e a sua menutenção. Foca-se especialmente na relação entre clientes, serviços prestados pela empresa e variáveis incluídas.

No programa, existem vários tipos de clientes. Existem os clientes não registados na empresa, que somente solicitam serviços rápidos (ficando os serviços solicitados guardados na empresa, mas não dados relativamente a clientes não registados);

Clientes Registados, que por sua vez podem assumir três formatos: Cliente Registado normal, Cliente Registado Empresarial, Cliente Registado Particular. Este tipo de clientes guarda em si um histórico de Serviços solicitados à empresa, que são usados mais tarde para o cálculo de possíveis promoções na cobrança dos serviços.

Não podem existir dois clientes idênticos, ou seja, com o mesmo id (que neste caso, é representado pelo seu número NIF).

Os pagamentos dos serviços podem também ser efectuados de vários tipos: aos clientes não registados estão à sua disposição os pagamentos por numerário e por multibanco. Aos clientes registados estão disponíveis todas e quaisquer formas de pagamento, variando estas entre numerário, multibanco, pagamento ao fim do mês e pagamento por cartão de crédito. De salientar que os pagamentos por cartão de crédito involvem um imposto de 5% do valor total do serviço e os pagamentos ao fim do mês involvem um imposto de 2% ao valor total de todos os serviços pagos no fim dest e período.

A empresa também poderá oferecer aos seus clientes registados certos descontos confirme estes tenham acumulado vários serviços solicitados à empresa no seu histórico de serviços.

Dois serviços são idênticos se possuirem a mesma origem, o mesmo destino e forem efectuados à mesma hora e data. Tal não pode suceder.

O valor de cada serviço é influenciado pela hora a que é feito (devido ao período de maior trânsito) e pela distância desse serviço.

Solução Implementada

Empresa.h/Empresa.cpp

A Empresa de Táxis em si mesma, componente principal do programa foi implementada usando uma classe Empresa que permite a gestão de seus clientes e serviços.

Cada empresa possui um nome e uma morada (ambos definidos por objectos do tipo std::string), não podendo duas empresas possuir um nome ou morada idênticos.

A classe Empresa possui em si os seguintes métodos públicos:

Empresa() {} -> constructor de um objecto do tipo Empresa com os seus membros inicializados a valor nulos

```
Empresa(std::string nome) –> constructor de um objecto Empresa com um nome específico
```

```
Empresa(std::string nome, std::string morada) -> constructor de um objecto Empresa co
um nome e uma morada específicos
```

```
std::string getNome() const -> retorna o nome da Empresa
```

```
std::string getMorada() const -> retorna a morada da Empresa
```

```
std::vector<Registado *> getRegistados() const -> retorna um vector que possui todosos
                            elementos Registados guardados pela Empresa
```

```
Registado* getRegistadoByNIF(unsigned int NIF) const -> retorna um cliente Registado
                                                        com um NIF específico, passado
                                                  como argumento
```

```
Servico* getServicoByID(unsigned int servID) const -> retorna um Serviço com um
Id específico, passado como argumento
```

```
bool adicionaCliente(Registado *cli) -> tenta adicionar um cliente Registado à
                                        Empresa, retornando true se conseguiu ou
                                    false se não conseguiu. Caso o Cliente
                                    seja repetido, lança uma excepção do tipo
                                    ClienteRepetido.

bool removeCliente(Registado *cli) -> tenta remover um cliente Registado da
                                        Empresa, retornando true se conseguiu ou
                                        false se não conseguiu. Caso o cliente
                                        solicitado no argumento da função não
                                        exista, lança uma excepção do tipo
                                        ClienteInexistente.

bool removeCliente(unsigned int NIF) -> tenta remover um cliente Registado com um NIF
                                        específico. Caso consiga, retorna true; caso
                                    não consiga, retorna false; Caso o cliente de NIF
                                    igual ao mandado como argumento da função não
                                    exista, lança uma excepção do tipo
                                    ClienteInexistente.

std::vector<Servico *> getHistorico() const -> retorna o vector de serviços guardados
                                                da empresa (histórico de serviços)

std::string getInformacao() const -> retorna toda a informação relativa à Empresa, já
                                        formatada, numa std::string. A informação
                                    consiste em toda a informação relativa a cada
                                    cliente e os seus respectivos históricos de
                                    serviços acumulados. Também indica o estado de
                                    cada serviço.

std::string getInformacaoServicos() const -> retorna toda a informação relativa a
                                                cada serviço acumulado pela Empresa.

std::string getNIFsClientes() const -> retorna uma std::string onde são escritos todos os
                                        NIF's de todos os clients registados na
                                    Empresa, formatados de uma forma amigàvel para o
                                    utilizador.

std::string getIDsServicos() const -> retorna uma std::string onde são escritos todos
                                        os Id's de todos os servicos de todos os
                                    clients guardados pela Empresa, formatados de uma
                                    forma amigável para o utilizador.

std::string getServicosByCliente(unsigned int NIF) const -> retorna uma std::string onde
                                                            são escritos todos
                                                        os Id's de um cliente registado
                                                        com um NIF específico,
```

formatados de uma forma amigável para o utilizador.

`bool adicionaServicoCliente(Servico* serv, unsigned int NIF)` -> adiciona um Serviço a um cliente com um NIF específico, retornando true caso consiga, false caso não consga, ou lançando uma excepção do tipo ClienteInexistente caso o cliente com o NIF passado como argumento da função não exista.

`bool servicoRapido(Servico* serv, Cliente* cli, std::string metodo_pagamento)` -> efectua um serviço rápido a um cliente não registao, retornando true caso consiga e false caso não consiga.

`bool pagamentoServicoCliente(unsigned int NIF)` -> efectua o pagamento de um service a um cliente com um NIF específico, retornando true caso consiga ou false caso não consiga.

As excepções ClienteRepetido e ClienteInexistente são definidas no ficheiro Empresa.h. ambas possuem os membros privados nome e moradda (definidos por uma std::string) e o membro NIF (definido por um unsigned int).

Possuem em si também os métodos ClienteInexistente() / ClienteRepetido(), ClienteInexistente(std::string nome, std::string morada) / ClienteRepetido(std::string nome, std::string morada) e ClienteInexistente(std::string nome, std::string morada, unsigned int NIF) / ClienteRepetido(std::string nome, std::string morada, unsigned int NIF) e ainda um método void printInfo() cada uma (imprime as informações relativas ao cliente defeituoso, de uma forma amigável ao utilizador).

Servico.h / Servico.cpp

Cada Serviço foi implementado recorrendo a uma classe serviço. Esta classe os seguintes objectos privados:

```
unsigned int id;
```

id do serviço, actualizado a cada criação de um novo serviço. Deste modo, o primeiro serviço criado possui id = 1, o segundo serviço criado possui id = 2, e assim successivamente. Esta funcionalidade foi conseguida recorrendo a variáveis estàticas.

```
std::string origem;
std::string destino;
```

Origem e Destino da viagem/service, respectivamente.

```
unsigned int distancia;
```

Distância da viagem/serviço, por agora não possui qualquer tipo de uso.

```
float valor;
```

Valor a pagar pelo serviço/preço do serviço. Depende da distância e hora a que é efectuado.

```
data_struct data_realizacao;
tempo horas;
```

Hora e data de realização de serviço, respectivemente. A hora é definida por uma struct (tempo) constituída pelos objectos horas, minutos, segundos; A data é definida Também por uma struct (data_realização) e é constituída pelos objectos dia, mês e ano. Todos os objectos das duas structs são definidos através de unsigned int

```
bool estado;
```

Estado do objectido, podendo variar entre “true” (pago) ou “false” (por pagar).

```
std::string tipo_pagamento;
```

Tipo de pagamento utilizado para pagar o serviço, podendo variar entre as std::string’s “numerario”, “cartao de credito”, “multibanco”, “fim do mes” para os clientes registados e pelas std::string’s “numerario” e “multibanco” para os clientes não Registados.

```
unsigned int NIF;
```

NIF do cliente que pagou o Serviço

Possui também os seguintes métodos públicos:

Servico() -> constructor de um Serviço com todos seus membros inicializados a valores Nulos

Servico(std::string origem, std::string destino) -> constructor de um Serviço inicializado ocm valores específicos de origem, destino

Servico(std::string origem, std::string destino, data_struct dt, tempo horas) -> constructor de um Serviço inicializado com valores específicos de origem, destino, horas e data

~Servico() -> destructor de um objecto Serviço

bool operator== (const Servico &serv) const -> operador== para dois serviços. Dois serviços são iguais se possuirem a mesma origem, o mesmo destino, a mesma hora e a mesma data.

std::string getOrigem() const -> retorna a origem de um serviço

std::string getDestino() const -> retorna o destino de um serviço

tempo getHoras() const -> retorna a hora de um serviço

float getValor() const -> retorna o valor/preço de um service

data_struct getData() const -> retorna a data de relização de um serviço

unsigned int getId() const -> retorna o Id de um destino

std::string getInformacao() const -> retorna toda a informação relativa a um serviço numa std::string, formatada de uma forma amigável ao utilizador

bool getEstado() const -> retorna o estado de pagamento de um serviço

unsigned int getDefaultId() const -> retorna o default_id, usado para a implementação de todos os restantes Id's de todos os serviços criados

float getDefaultValor() const -> retorna o valor genérico de um serviço, sem qualquer tipo de imposto acrescido

tempo getInicioPeriodoTransito() const -> retorna o inicio do periodo de maior trânsito

```
  tempo getFimPeriodoTransito() const -> retorna o fim do período de maior trânsito
float getImpostoTransito() const -> retorna o imposto calculado ao preço de um
                                    serviço

  unsigned int getNIF() const -> retorna o NIF do cliente que pagou o serviço


  void setOrigem(std::string origem) -> define a origem de um serviço

  void setDestino(std::string destino) -> define o destino de um serviço

  void setData(data_struct dt) -> define a data de realização de um serviço

  void setHoras(tempo horas) -> define a hora de realização de um serviço

  void setPago() -> define um serviço como pago

  void setTipoPagamento(std::string tipo) -> define o tipo de pagamento do serviço

  void setValor(float valor) -> define o valor de um serviço

  void setNIF(unsigned int NIF) -> define o NIF do cliente que pagou o serviço

  void setPeriodoTransito() -> define o period de maior hora de trânsito

  void setImpostoTransito(float imposto) -> define o imposto acrescido ao valor de um
                                            serviço
```

Cliente.h / Cliente.cpp

Os clientes foram implementados recorrendo a técnicas de polimorfismo. Deste modo, existe uma class mãe Cliente de onde a qual são herdados os difernetes tpo s de clientes.

Da classe mãe Cliente é herdada a classe Registado (cliente Registado);

Da subclasse mãe Registado são herdados os clientes Empresariais (Empresarial) e os clientes particulares (Particular).

A classe mãe Cliente possui os seguintes membros privados (herdados aos outros tipos de clientes):

```
std::string nome;

nome do cliente
```

```
std::string morada;

morada do cliente
```

```
unsigned int NIF;

NIF do cliente (serve como identificaro de cada cliente, já que cada cliente possui
um NIF único)
```

```
A classe mãe Cliente possui os seguintes métodos públicos:
```

```
Cliente() -> constructor do default Cliente (inicializado com todos os seus
             membros privados a valores nulos)
```

```
Cliente(const std::string nome , const std::string morada, const unsigned int NIF) ->

Constructor de um Cliente com um nome, morada e NIF específicos
```

```
Cliente(const std::string nome, const std::string morada) -> constructor de um
                                                              Cliente com nome e
                                                          morada específicos

Cliente(std::string nome) -> constructor de um Cliente com nome específico

~Cliente() -> destructor de um objecto do tipo Cliente

bool operator== (const Cliente &cli) const -> operador == para dois clients. Dois
                                                 clients só são idênticos se possuirem
                                            o mesmo id (neste caso, o número NIF)

bool operator< (const Cliente &cli) const -> operador < para dois clients. Um cliente é
                                                 "menor" que outro cliente se
                                            possuir um nome de maior grandeza,
                                            considerando-se a ordem alfabética de
                                            organização dos clientes, pelo seu nome

void operator= (const Cliente &cli) -> operador = para dois clients. Coloca os

                                            valores de nome, morada e NIF de um
                                       cliente nos respectivos valores correspondentes
                                       de outro cliente.

//METODOS GET

std::string getNome() const -> retorna o nome de um cliente

std::string getMorada() const -> retorna a morada de um cliente

unsigned int getNIF() const -> retorna o NIF de um cliente

virtual std::string getInformacao() const -> função virtual, retorna toda a informação

                                                 relativa a um cliente não registado,
                                            formatada numa string de uma forma amigável
                                            ao utilizado

//METODOS SET

void setNome(std::string nome) -> define um nome para um cliente

void setMorada(std::string morada) -> define uma morada para um cliente

void setNIF(unsigned int NIF)  →  define um NIF para um cliente
```

```
//METODOS RELATIVOS A SERVICOS

virtual bool pagamentoNumerario(Servico &serv) -> função virtual que efectua o
                                                  pagamento de um serviço pelo método
                                                  de pagamento numerário

virtual bool pagamentoMultibanco(Servico &serv) -> função virtual que efectua o
                                                       pagamento de um serviço pelo
                                                  método de pagamento por multibanco
```

Clientes Registados

A classe Regsitados, herdada da classe mãe Cliente, possui os seguintes membros privados adicionais:

```
  std::vector<Servico *> historico;
```

```
Vector que guarda todos os (apontadores para) serviços que um cliente
Registado solicitou à Empresa.
```

Possui os seguintes métodos públicos:

```
Registado() –> constructor de uma class default Registado com todos os
               membros privados inicializados a valores nulos

  Registado(std::string nome, std::string morada, const unsigned int NIF) ->
                     constructor de um Registado com nome, morada e NIF específicos

  Registado(std::string nome, std::string morada) -> constructor de um Registado com
                                                     nome e morada específicos

  Registado(std::string nome) -> constructor de um Registado com nome específico
```

```
std::vector<Servico *> getHistorico() const -> retorna o vector de apontadores ara
                                                serviços de um cliente Registado


  virtual std::string getInformacao() const -> retorna toda a informação relativa a um
                                                Registado numa std::string, formatada
                                           de forma amigável ao utilizador


  //METODOS RELATIVOS A SERVICOS

  void operator=(const Registado &reg) -> operador = para classes do tipo Registado;
                                           dois Registados são iguais se possum o mesmo
                                      valor de NIF


  virtual std::string getInformacaoServicos() const -> retorna toda a informação
                                                        relativa aos serviços guardados
                                                    no vector "histórico" de um Registado
                                                    numa std::string, formatada de uma
                                                    forma amigável para o utilziador


  virtual std::string getTipo()const -> retorna o tipo de cliente (neste caso,
                                           "registado")


  virtual bool pagamentoNumerario(Servico &serv) -> função virtual derivada da função
                                                        de igual nome da classe mãe
                                                    Cliente. Efectua o pagamento de um
                                                    serviço através do método de
                                                    pagamento por numerario.


  virtual bool pagamentoMultibanco(Servico &serv) -> função virtual derivada da função da
                                                        classe mãe Cliente. Efectua
                                                    o pagamento de um serviço através do
                                                    método de pagamento por multibanco.


  virtual bool pagamentoCartaoCredito(Servico &serv) -> função virtual que efectua o
                                                        pagamento de um serviço pelo
                                                    método de "cartao de credito",
                                                    acrescendo ao valor total do
                                                    serviço um imposto de 5%
```

```
virtual bool pagamentoFimDoMes(std::vector<Servico *> servs)
```
-> função virtual que efectua o pagamento dos vários serviços passados como argumento da ufnção num vector de apontadores para serviços pelo método de pagamento “fim do mes”, acrescendo ao valor total de todos os serviços um imposto de 2%

O pagamento só é efectuado se as datas de pagamento dos vários serviços se encontrarem entre o primeiro dia do mês passado e o último dia do mês actual.

```
virtual bool pagamentoNumerario(unsigned int id)
```
-> função virtual derivada da função da classe mãe de mesmo nome que efectua o pagamento de um serviço com Id específico pleo método de pagmaento por “numerario”

```
virtual bool pagamentoMultibanco(unsigned int id)
```
-> função derivada da função da classe mãe de mesmo nome que efectua o pagamento de um serviço com um Id específico pelo método de pagamento por “multibanco”

```
virtual bool pagamentoCartaoCredito(unsigned int id)
```
-> função virtual que efectua o pagamento de um serviço com um Id específico pelo método de pagamento

por “cartao de credito”, acrecendo ao valor total do serviço um imposto de 5%

```
virtual bool pagamentoFimDoMes(std::vector<unsigned int> ids)
```
-> função virtual que efectua o pagamento de vários serviços, cada um com um Id específico passados como argumento num vector de Id’s pelo método de pagmaento “fim do mês”. O pagamento só é efectuado se as datas de pagamento dos vários serviços se encontrarem entre o primeiro dia do mês passado e o último dia do mês actual.

NOTA: para todas as funçoes de pagamento, é retornado o valor true caso o pagamento seja feito com successo ou false caso o pagamento tenha falhasdo.

`bool adicionaServico(Servico* serv)` -> adiciona um Serviço a um cliente Registado. Retorna true caso consiga e false caso não consiga.

Clientes Empresariais

As classe Empresarial é derivada da sub-classe mãe Registado.

Não apresenta nenhuns valores privados adicionais.

Possui os seguintes métodos públicos:

`Empresarial()` -> constructor de um objecto defaulte Empresarial, inicializao com valores nulos.

`Empresarial(std::string nome, std::string morada, const unsigned int NIF)` -> constructor de um objecto Empresarial, incializado com valores específicos de nome, morada e NIF.

`Empresarial(std::string nome, std::string morada)` -> constructor de um objecto Empresarial, inicializado com valores específicos de nome e morada.

`Empresarial(std::string nome)` -> constructor de um objecto do tipo Empresarial com um nome específico

`virtual std::string getInformacao() const` -> função virtual derivada da função de mesmo nome da sub-classe mãe Registado que retorna toda a informação relativa ao Empresarial numa std::String, formatada de forma amigável ao utilizador.

`virtual std::string getTipo() const` -> função virtual que retorna o tipo de Registado que o objecto é (neste caso, retorna “emrpesarial”)

Clientes Particulares

Tal como o objecto Empresarial, a classe Particular é derivada da sub-classe mãe Registado, e também não possui métodos privados adicionais.

Possui os seguintes métodos públicos:

```
Particular() -> constructor de um objecto default Particular, inicilaizado com
                                        todos os valores a nulo

Particular(std::string nome, std::string morada, const unsigned int NIF) ->

              constructor de um objecto Particular com nome, morada e NIF específicos

Particular(std::string nome, std::string morada) -> constructor de um objecto
                                        Particular com nome e omrada específicos

Particular(std::string nome) -> constructor de um objecto Particular co nome
                                    Específico

virtual std::string getInformacao() const -> função virtual derivada da função de memo
                                                nome da sub-classe mãe Registado que
                                        retorna toda a informação relativa ao
                                        objecto Paritcular, num std::string
                                        formatada de forma amigável ao utilizador.

virtual std::string getTipo() const -> função virtual que retorna o tipo de Registado
                                    que o objecto é (neste caso, retorna "particular")
```

Interface.h / Interface.cpp

A interface é o meio de comunicação entre o utilizador e o programa. É implementada através de um classe Interface.

Não possui nenhuns membros privados, só métodos públicos, já que somente tem a função de manusear o programa.

Deste modo, a classe Interface possui os seguintes métodos declaradaos na classe:

```
int GetInput() -> solicita ao utilizador um integral e retorna esse integral. Usado
                    na escolha das opções, em cada menu.

void DisplayMenuInicial() -> imprime o Menu Inicial do programa

void MenuInicial(Empresa* emp) -> chama o Menu Inicial do programa. Neste menu, são
                                  assinaladas quaisquer excepções vindas do ficheiro
                             main.cpp e o programa comprimenta o utilizador. Imprime
                             uma mensagem de terminação de programa quando este retorna
                             desta função.

void DisplayMenuPrincipal() -> imprime o Menu Principal do programa

void MenuPrincipal(Empresa* emp) -> chama o Menu Principal do programa, tomando como
                                      argumento a Empresa do utilizador em questão.
                                   Nest menu, o utilziador tem as seguintes opções:

0 - Informacao geral -> imprime toda a informação relativa à Empresa em questão,
                          de uma forma amigável ao utilziador

1 - Informacao relativa aos Servicos -> imprime toda a informação relativa a todos os
                                        serviços de todos os clientes da Empresa em
                                   questão, de uma forma amigável ao utilziador
```

2 - Efectuar servico rapido (para clientes nao registados) -> invoca uma função serviçoRápido() que permite efectuar um serviço a um cliente não registado e guardá-lo na Empresa, sem guardar dados sobre o cliente que o efectuou

3 - Adicionar Registado (cliente registado) -> adiciona um Registado à Empresa

5 - Adicionar Servico a Cliente -> adiciona um service a um cliente

6 - Pagamento de Servicos -> invoca o menu de pagamento de serviços

7 - Carregar ficheiro -> permite carregar um programa a partir de ficheiros

8 - Guardar sessao actual -> permite guardar em ficheiros o programa actual

9 – Sair -> retorna ao Menu Inicial

```
void NovoFicheiro(Empresa* emp) -> cria um novo ficheiro
```

```
void CarregaFicheiro(Empresa* emp) -> carrega um programa novo a partir de ficheiros. Os Clientes são carregados de um ficheiro [nome_d_ficheiro]_clientes.txt; os Serviços são carregados de um [nome_do_ficheiro]_servicos.txt; A Empresa carregada de um ficheiro [nome_do_ficheiro]_empresa.txt.
```

```
void GuardaFicheiro(Empresa* emp) -> guarda o ficheiro actual em ficheiros. Os Clientes são guardados num ficheiro [nome_d_ficheiro]_clientes.txt; os Serviços são guardados num ficheiro [nome_do_ficheiro]_servicos.txt; A Empresa é guardada num ficheiro [nome_do_ficheiro]_empresa.txt.
```

```
Cliente* CriarCliente() -> invoca o menu de criação de um Cliente, retornando-o.
```

```
Registado* CriarRegistado() -> invoca o menu de cração de um Registado, retornando-o.
```

```
Servico* CriarServico() -> invoca o menu de criação de um Serviço, retornando-o.
```

```
Empresa* CriarEmpresa() -> invoca o menu de criação de uma Empresa, retornando-a.
```

```
void DisplayMenuPagamento() -> imprime o Menu de Pagamento de uma forma amigável
                               ao utilizador.


void MenuPagamento(Empresa* emp) -> chama o Menu de Pagamento. Neste, menu, o

                                    utilizador escolhe de entre os quatro tipos de
                                pagamentos, escolhe um cliente, e escolhe um dos
                                serviços desse cliente para pagar.
```

NOTA: no ficheiro Interface.cpp existem também os seguintes métodos, declarados fora da class Interface:

```
void Pausa() -> para o programa até o utilizador carregar numa tecla e chama a função
                clearBuffer(). Semelhante ao system("pause").

void clearBuffer() -> limpa o input buffer.
```

Utilidades.h

Neste ficheiro estão declarados métodos e objectos auxiliaries que são usados pelo resto do programa.

Estão neste ficheiro declarados os seguintes objectos:

```
struct tempo{
	unsigned int horas;
	unsigned int minutos;
	unsigned int segundos;
};
```

Struct para definição da hora a que um Serviço é realizado.

```
struct data_struct{
	unsigned int dia;
	unsigned int mes;
	unsigned int ano;
};
```

Struct para a definição da data a que um Serviço é realizado.

BST.h

Ficherio fornecido nas aulas práticas de AEDA. Possui uma árvore binária e métodos adicionais para o seu funcionamento.

ServicoPointer.cpp/ServicoPointer.h

Estes ficheiros possuem funcionam como um intermediário entre a classe Serviço e a BST. Uma Empresa passa a possuir como membro adicional uma BST de objetos do tipo ServicoPointer. Cada ServicoPointer corresponde a um Servico:

```
Servico* servicoPtr
```

Pointer para o Servico a que este objeto ServicoPointer corresponde.

Esta classe possui os seguintes métodos:

```
ServicoPointer(Servico* pointer) -> constructor de um objeto ServicoPointer. Aceita
                            como argumento um Servico, ao qual corresponderá, na BST.
```

```
void setPointer(Servico* pointer) -> função que altera o Servico correspondente
                                    de um ServicoPointer; aceita como argumento o novo
                            Servico correspondente
```

```
Servico* getPointer() const -> função que retorna o Servico correspondente do
                                    ServicoPointer
```

```
bool operator<(const ServicoPointer &rRHS) const -> overload do operador < para
                                                    objetos ServicoPointer; um
                                                    ServicoPointer1 é menor que outro se o
                                                    nome do Servico correspondente for de
                                                    ordem alfabética menor que o nome de
                                                    um objeto ServicoPointer2 passado como
                                                    argumento da função (objeto rRHS)
```

```
bool operator==(const ServicoPointer &rRHS) const -> overload do operador == para
                                                        objetos ServicoPointer; um
                                                    ServicoPointer1 é igual a outro se
                                                    possuírem Serviços correspondentes
                                                    iguais (uso do operador == para
                                                    objetos do tipo Serviço)
```

RegistadoPointer.cpp/RegistadoPointer.h

Estes ficheiros possuem funcionam como um intermediário entre a classe Registado e a Hash Table. Uma Empresa passa a possuir como membro adicional uma Hash Table de objetos do tipo RegistadoPointer, representando a lista de clientes inativos. Um cliente considera-se inativo quando o último serviço efetuado foi realizado no ano passado. Cada RegistadoPointer corresponde a um Registado:

```
Registado* RegistadoPtr
```

Pointer para o Registado a que este objeto RegistadoPointer corresponde.

Esta classe possui os seguintes métodos:

```
RegistadoPointer(Registado* pointer) -> construtor de um objeto RegistadoPointer.
                                        Aceita como argumento o Registado pointer a
                                    que passará a corresponder

void setPointer(Registado* pointer) -> função que altera o Registado a que este objeto
                                        RegistadoPointer corresponde. Aceita como
                                    argumento o Registado a que passará a corresponder

Registado* getPointer() const -> função que retorna o Registado a que este objeto
                                    RegistadoPointer corresponde

bool operator<(const RegistadoPointer &rRHS) const -> overload do operador < para
                                                      objetos do tipo
                                                  RegistadoPointer; um RegistadoPointer1
                                                  é menor que outro quando o seu nome é
                                                  de ordem alfabbética menor que o outro
                                                  RegistadoPointer2, passado como
                                                  argumento da função (objeto rRHS)

bool operator==(const RegistadoPointer &rRHS) const -> overload do operador == para
                                                       objetos do tipo
                                                   RegistadoPointer; dois objetos
                                                   RegistadoPointer são iguais se os seus
                                                   Registados correspondentes forem
                                                   iguais (uso do operador == para
                                                   objetos do tipo Registado)
```

Empresa.cpp/Empresa.h – 2ª Parte

A partir da 2ª do projeto, a classe Empresa passa a possuir estes membros dados adicionais:

```
BST<ServicoPointer*> arvore_servicos
```

Árvore binária de pesquisa que guarda todos os pointers para objetos do tipo ServicoPointer, cada um correspondente a um Serviço. Suncintamente, fornece acesso de um modo indireto a todos os Serviços (ou seja, faturas) da empresa.

```
tabHCliente tabela_clientes_inativos
```

Tabela de dispersão que guarda em si todos os pointers para objetos do tipo RegistadoPointer, cada um correspondente a um Registado. Suncintamente, fornece acesso de um modo indireto a todos os clientes Registados na empresa que se encontram inativos, ou seja, cujo último serviço efetuado foi realizado no ano passado (com base na macro ANO_ACTUAL, definido no ficheiro Empresa.cpp).

Esta classe possui os seguintes métodos novos, a partir da 2ª parte do projeto:

Relativamente a BST's:

```
BST<ServicoPointer*> getArvoreServicos() const -> retorna a BST arvore_servicos da
                                                  Empresa

bool addServicoBST(Servico* serv) -> adiciona um Servico serv passado como argumento
                                     da função à BST arvore_servicos da empresa

void printArvoreServicos() const -> imprime toda a informação relativamente a todos os
                                    elementos da BST arvore_servicos da empresa

void updateArvoreServicos() -> atualiza a BST arvore_servicos da empresa, colocand
                               nela os Servicos entretanto criados que ainda não se
                               encontravam na BST, e copiando todos os Serviços do vetor
                               de servicos "historico" da empresa se a BST se encontra
                               vazia
```

Relativamente a Hash Table's:

```
tabHCliente getTabelaClientesInativos() const -> retorna a Hash Table
                                                 tabela_clientes_inativos da empresa

void printTabelaClientesInativos() const -> imprime informação sobre todos os
                                                 elementos da Hash Table
                                            tabela_clientes_inativos da empresa

bool updateTabelaClientesInativos() -> atualiza a Hash Table tabela_clientes_inativos
                                          da empresa; percorre o vetor de clientes
                                    Registados "clientes" da empresa e caso o último
                                    serviço realiazdo por um destes datar do ano
                                    passado, retira copia esse cliente e insere-o na
                                    Hash Table; percorre todos os elementos da Hash
                                    Table e, caso o último serviço realizado por um
                                    destes datar do ano atual, remove-o da Hash Table
```

Interface.cpp/Interface.h – 2ª Parte

Table A patir da 2ª parte do projeto, foi criado um novo menu relativamente às opções que as novas estruturas de dados oferecem. Esse menu é acedido através de uma nova opção acrescentada à função "MenuPrincipal", intitulada "Opções das Estruturas de Dados".

Deste modo, criaram-se as seguintes funções:

```
void DisplayMenuEstruturasDados() -> imprime as opções oferecidas pelo Menu Estrura
                                          Dados

void MenuEstruturasDados(Empresa* emp) -> chama o Menu Estrura Dados do programa,
                                          tomando como argumento a Empresa do utilizador
                                          em questão.
                                    Nest menu, o utilziador tem ao seu dispor as
                                    seguintes opções:
```

0 - Informacao da BST de servicos -> invoca a função da empresa printArvoreInfo(), imprimindo toda a informação sobre a BST de serviços da empresa, organizados pelo nome do cliente correspondente

1 - Actualiza arvore de servicos realizados -> invoca a função da empresa updateArvoreServicos(), atualizando a árore de serviços da empresa

2 - Informacao sobre clientes inativos (Hash Table) -> invoca a função da empresa printTabelaClientesInativos(), imprimindo toda a informação sobre os clientes atualmente inativos da empresa

3 - Actualiza tabela de clientes inativos -> invoca a função da empresa updateTabelaClientesInativos(), atualizando a tabela de clientes atualmente inativos da empresa

4 - Informacao sobre clientes em fila de espera (Priority Queue) ->

5 - Actualiza fila de clientes em espera ->

6 – Sair -> sai do Menu Estrutura Dados e retorna para o Menu Principal do programa

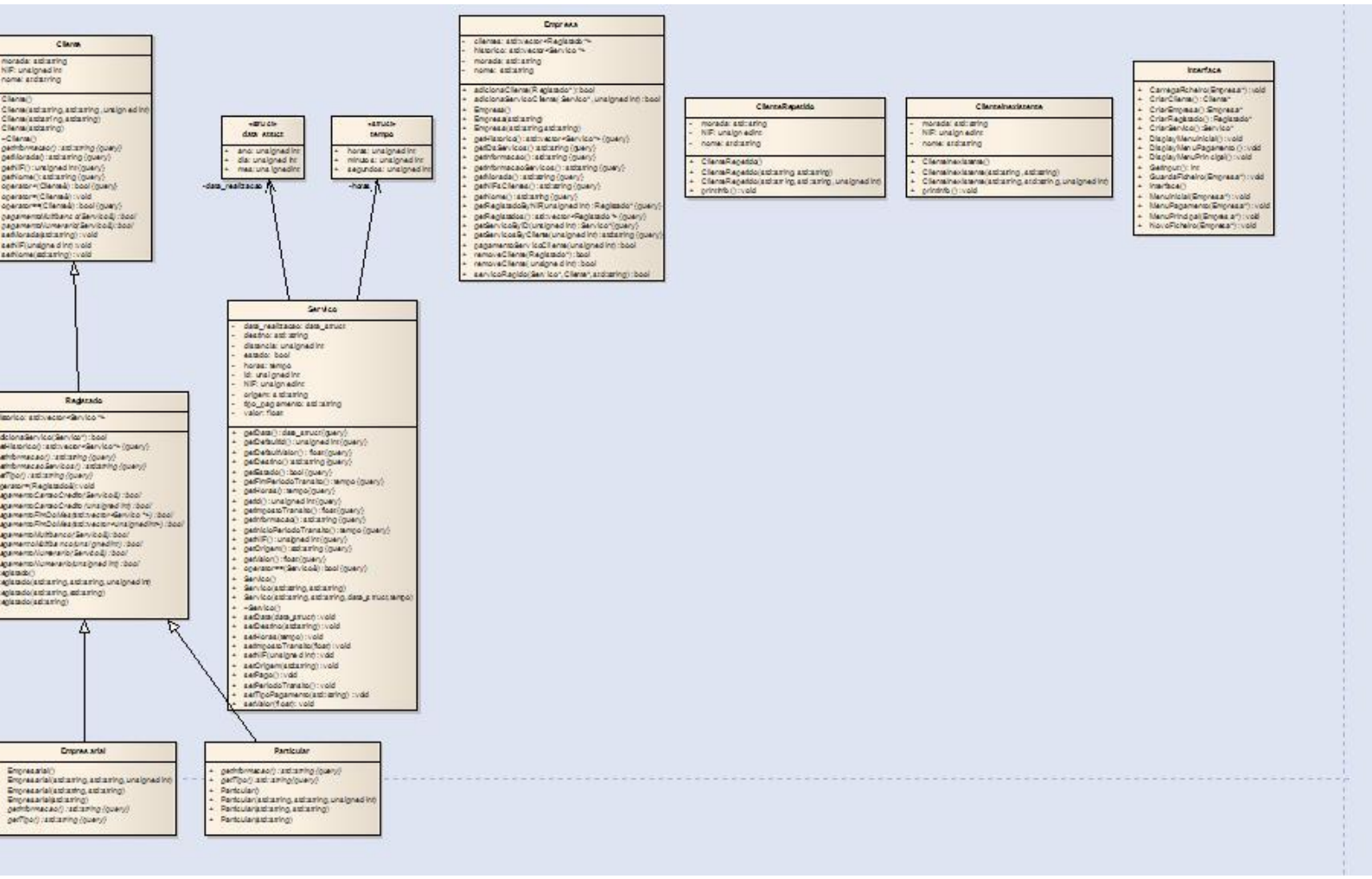
Main.cpp

O programa é executado pelo ficheiro main.cpp. Neste ficheiro foram feitos os diversos testes do programa, incluindo tratamento de excepções e declaração de objectos.

É neste ficheiro que é criada e invocada a class Interface.

O programa inicializa ao solicitar ao utilizador a hora de maior trânsito.

Diagrama de Classes (UML)



Casos de Utilização

2Criação de Empresas de Táxis, ou qualquer tipo de Empresa relacionada com transportes

Alterar nome da Empresa

Adicionar Clientes à Empresa

Adicionar Serviços aos Clientes da

Empresa Remover Clientes da Empresa

Criação de Serviços

Criação de Clientes

Impressão de informação relacionada com a Empresa

Impressão de informação relacionada com os

Clientes Impressão de informação relaiconada com

os Serviços Entre outros

Dificuldades Encontradas

A dificuldade principal foi criar uma estrutura segura para o programa. Devido à falta de tempo, o progama não está perfeitamente estável e a estrutura geral podia ser melhorada. Também devido a esse factor não foi possível implementar todas as funcionalidades pretendidas do programa.

Foram encontradas dificuldades ao nível do IDE Eclipse, com o qual o projecto foi primeiramente realizado. Devido a problemas com esse IDE que não foram possíveis de ser soluciconados, foi usado o IDE Visual Studio Enterprise 2015, fornecido pelo Dremaspark.

O resultado final não foi o esperado, já que pode-se considerar o trabalho “incompleto”, devido à falta de tempo.

Distribuição de Trabalho pelos Elementos do Grupo

Antes da inicialização de distribuição de trabalho pelo grupo, os dois restantes membros desistiram da cadeira. Devido a este precalce, todo o trabalho, quer na 1ª parte quer na 2ª parte do projeto, foi efectuado pelo aluno Nuno Manuel Ferreira Corte-Real.