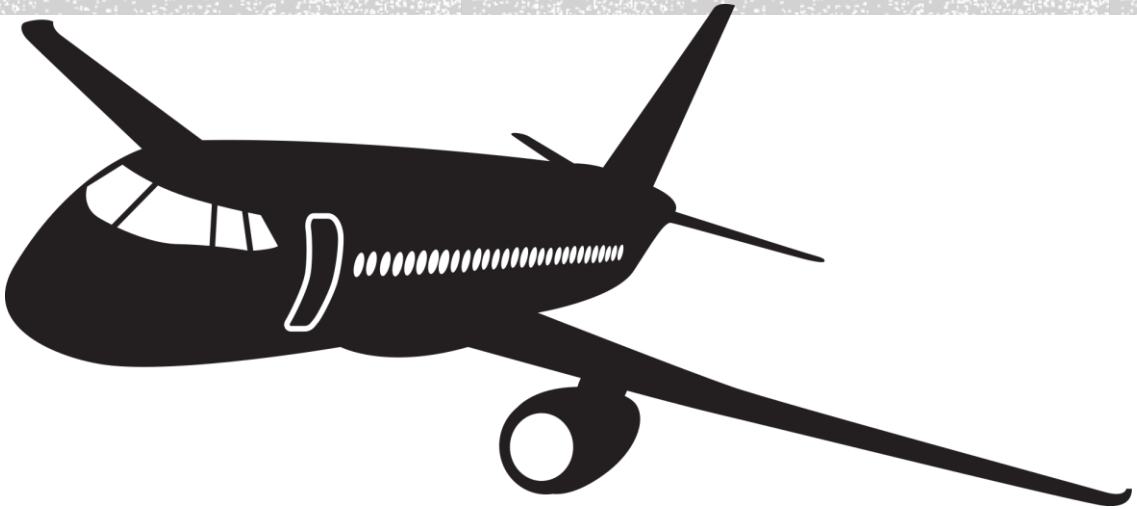
ALGORITMOS E ESTRUTURAS DE DADOS

Gestão de informação em uma companhia aérea

Grupo 12



José Sousa 202006141
Bernardo Campos 202006056

DESCRIÇÃO DO PROBLEMA

A gestão de informação em uma companhia aérea requer a utilização de uma série de métodos de recolha, armazenamento e distribuição de informação.

Para além de guardar a informação acerca da companhia aérea em estruturas lineares, é necessário fazê-lo de modo a que a mesma seja mantida mesmo após o término do programa.

SOLUÇÃO DO PROBLEMA

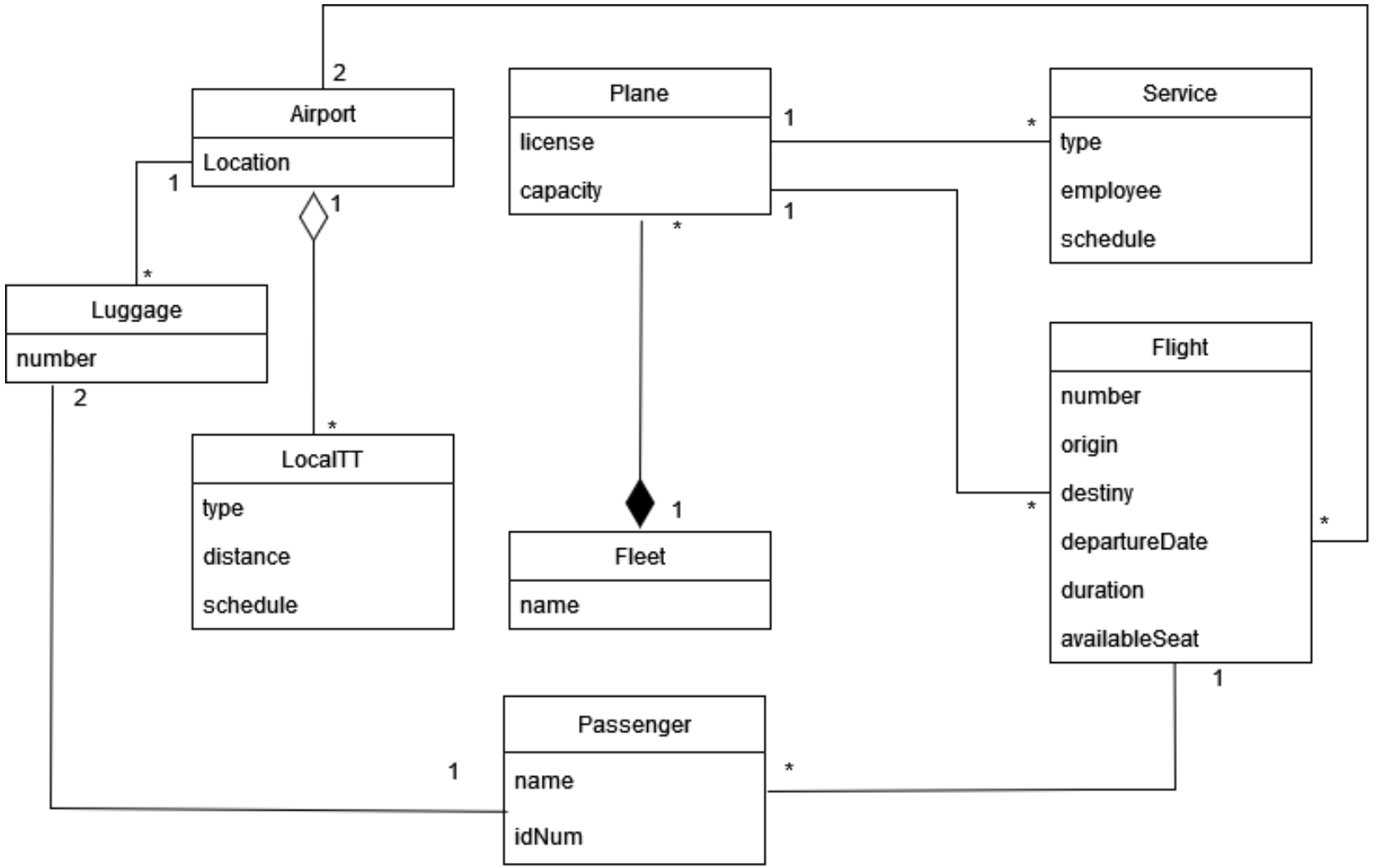
-Utilização de toda a informação inserida até ao momento: escrita e leitura de ficheiro de texto (.txt)

-Acesso a informação guardada em estruturas lineares

-''Update'' das estruturas lineares com a informação guardada nos ficheiros de texto (sempre que o programa inicia)

-Implementação de algoritmo que permita a detecção e não processamento de informação repetida, pois esta poderá por em causa pesquisas ou leituras.

DIAGRAMA UML



ESTRUTURA DE FICHEIROS

flights.txt

- matrícula do avião
- número do voo
- capacidade
- origem
- destino
- data do voo

```
a123 300 120 porto lisbon 30/12/2021 14h
g123 8917 120 madrid lisbon 21/12/2021 18h
g123 8812 120 kyoto lisbon 21/11/2021 18h
```

transports.txt

- cidade
- tipo de tranporte
- distancia (km)
- horário (hora - minuto)

```
porto subway 5 18 30
lisbon bus 1 17 15
madrid train 2 8 30
```

planes.txt

- matrícula
- capacidade

```
a123 500
b170 200
c190 100
g123 900
p123 100
```

services.txt

- matrícula
- tipo de serviço
- responsável
- data

```
a123 cleaning pedro 22 / 10 / 2022 19h
g123 maintenance luis 9 / 1 / 2022 8h
a123 cleaning bernardo 24 / 12 / 2021 23h
a123 maintenance henrique 25 / 12 / 2021 8h
```

airports.txt

-cidade

```
porto
lisbon
madrid
tokyo
```

availableTickets.txt

- número do voo
- número de lugares diponíveis

```
300 150
8917 900
8812 900
```

passengers.txt

- número do voo
- ultimo nome
- número do CC

```
300 sousa 30879566
300 ferreira 15679845
300 campos 12345678
```

PESQUISA DE INFORMAÇÃO

Estruturas lineares

```
bool Management::checkServices(string license, string type, string employee, Date date){
    int i=0, j=0, plane;

    for(auto p: f1.getFleet()){
        if(p.getLicense() == license){
            if(p.getServices().size()==0) return false;
            for(auto it = p.getServices().begin();it!=p.getServices().end();++it){
                if((*it).getType()==type && (*it).getEmployee()==employee && (*it).getDate().year==date.year
                    return true;
                }
            }
        }
    }
    return false;
}
```

Ficheiro de texto

```
bool Management:: searchFileFlight(string num){
    ifstream f;
    f.open( s: "flights.txt", ios::in);
    string read;

    while(f>>read){
        if(num == read){
            return true;
        }
    }
    f.close();
    return false;
}
```

ESCRITA DE INFORMAÇÃO

```
ofstream f;
f.open( s: "services.txt", ios::app);
f<< license <<" "<< type << " " << employee << " " << date.day << " / " <<date.month << " / " <<date.year << " " << date.hour << "h\n";
f.close();
```

```
ofstream f;
f.open( s: "passengers.txt", ios::app);

f << fNum << " " << n << " " << i << "\n";
f.close();
```

```
ofstream f;
f.open( s: "transports.txt", ios::app);
f<< airport << " " << type << " " << distance << " " << hour << " " << minute << "\n"
f.close();
```

Estado: OK

UPDATE DE INFORMAÇÃO

```
//dá update ao vector de planes, caso já haja avioes registados no ficheiro
void Management::updatePlanes() {
    ifstream f;
    f.open( s: "planes.txt",ios::in);
    bool exists = false;

    string license;
    int cap;

    while(f>>license>>cap){
        for(auto p1:f1.getFleet()){
            if (p1.getLicense() == license){
                exists = true;
            }
        }
        if(not exists) {
            Plane p( license: license, cap);
            f1.addPlane( p: p);
        }
    }
}
```

Estado: OK

LEITURA DE INFORMAÇÃO

```
ifstream f;
f.open( s: "airports.txt", ios::in);
string location;

while(f>>location){
    if(checkAirport( location: location)== false){
        Airport airport( loc: location);
        airports.push_back(airport);
    }
}
```

```
while(f>>airport>>type>>distance>>hour>>minute){

    if(!checkTransport( airport: airport,  type: type, distance, hour, minute)){
        LocalTT local( t: type, distance, hour, minute);
        //find airport
        int i=0;
        while(i<airports.size()){
            if(airports[i].getLocation()==airport){
                airports[i].addTransports( t: local);
            }
            i++;
        }
    }
}
```

Estado: OK

LISTAGEM DE INFORMAÇÃO

```
void Management::updateServices(){
    ifstream f;
    f.open( s: "services.txt",ios::in);

    string type, employee, license;
    Date date;
    char sep;

    while(f>>license>>type>>employee>>date.day>>sep>>date.month>>sep>>date.year>>date.hour>>sep){
        for(auto &p:f1.getFleet()){
            if(p.getLicense() == license){
                Service service( t: type, date, f: employee, m: license);
                p.addServicelist( &: service);
                p.sortServicelist();
                p.listtoqueueService();
            }
        }
    }
}
```

Estado: OK

COMPRA DE BILHETES

-É provavelmente a feature com as funções mais completas e complexas do programa.

-Envolve o registo de passageiro e a entrada de bagagem no sistema de check in automático

-Requere a utilização de 5 classes diferentes

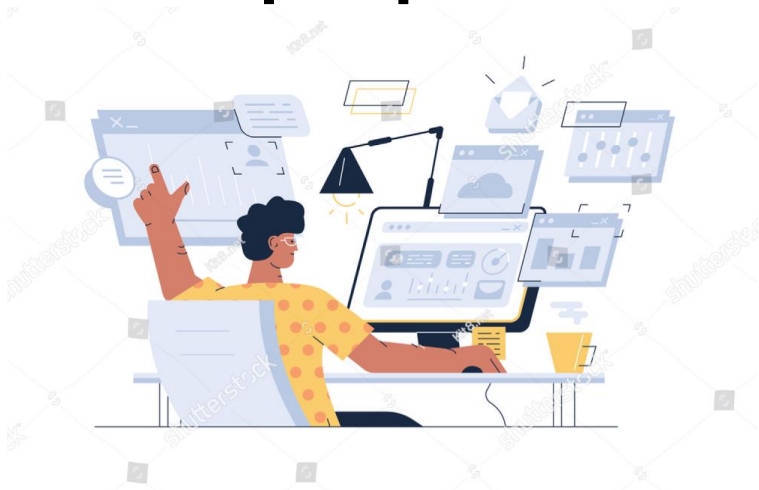
```
/**
* @brief Asks the user for the desired airports of origin and destiny, displays different flights,
* allows the user to buy tickets for those flights
*/
void userSearchFlight();

/**
* @brief Used in the beginning of the program, updates the number of seats available for booking in flights
*/
void updateAvailableSeats();

/**
* @brief Checks if there are any flights that match the user's request (origin, destiny, and number of tickets);
* if there are available flights, returns true and prints them; otherwise, returns false
* @param ori is the origin
* @param des is the destination
* @param nTickets is the number of tickets that the user wants to book
 * @return bool
*/
bool flightAvailability(string ori, string des, int nTickets);
```

EXEMPLO DE EXECUÇÃO DA COMPRA DE BILHETES

DIFICULDADES E DIVISÃO DE TRABALHO

-Actualização de listas e vectores, lendo informação de ficheiros: Numa fase inicial do trabalho, não estavamos a conseguir perceber o porquê do código não estar a funcionar correctamente, e perdemos bastante tempo a investigar e tentar perceber o que estava mal.

-A compra de bilhetes foialgo que nos deu bastante trabalho; tendo em conta a abordagem que tivemos do trabalho, a contrução de todo o algoritmo de compra de bilhetes envolveu a utilização de quase todas as classes que compõe o trabalho



O trabalho foi bem dividido entre nós, e cada um cumpriu aquilo que lhe foi designado.

- STRING S = “FIM!”

STD::COUT << S << ‘\N’;