PetWalker

Empresa de passeadores e cuidadores de cães (Parte 1)

Turma 6 – Grupo 2

Ana Matilde Guedes Perez da Silva Barra – up201904795
Maria Beatriz Russo Lopes dos Santos - up201906888
Ricardo Filipe da Costa Cabral Ferreira - up201907835

Descrição do Problema

Implementar funções que permitissem formar a empresa constituída por funcionários e clientes.

A empresa tem como objetivo o bem estar de animais, neste caso, cães, possuindo vários serviços para os mesmos (levar ao veterinário, tomar banho, cortar o pelo, passear e aulas de obediência).

Na parte dois do projeto foi nos pedido para implementar três estruturas de dados não lineares:

- **Árvore de pesquisa binária:** guarda todos os cães que já tenham realizado pelo menos que um serviço;
- **Fila de prioridade:** guarda as disponibilidades dos empregados, ficando no topo o que tem disponibilidade mais cedo;
- **Tabela de dispersão:** guarda os utentes que já não usam o serviço a mais de um dado tempo (1 mês).

Descrição da Solução

Quatro classes essenciais:

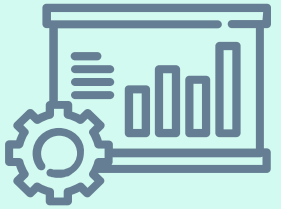
- Classe para os **clientes** (inclui todos os dados do cliente);
- Classe para os **funcionários** (inclui todos os dados do funcionário, bem como o dinheiro ganho);
- Classe para os **serviços** (cria um serviço com o cliente, o funcionário, a data prevista e o tipo de serviço que é desejado);
- Classe **PetWalker** (‘main’ classe que compila todas as outras classes).

Criamos mais uma:

- Classe para os **cães** (inclui todos os dados do cão, bem como o último serviço realizado por este);

Para uma melhor organização, concordamos em criar, para além da fila de prioridade referida anteriormente, uma outra para cada funcionário, tendo como objeto as datas disponíveis por este.

Algoritmos relevantes

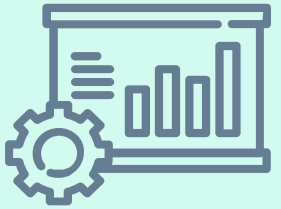


```
bool Dog::operator < (const Dog &d1) const {
    if (race == d1.race) {
        if (name == d1.name) {
            if (birth == d1.birth) {
                return getOwnerName() < d1.getOwnerName();
            }
            return birth < d1.birth;
        }
        return name < d1.name;
    }
    return race < d1.race;
}
```

```
struct OldClientsRecordsHash {

    int operator()(const OldClientsRecords& ar) const {
        int v = 0;
        for (unsigned int i = 0; i < ar.getEmail().size();i++)
            v = 37*v+ar.getEmail()[i];
        return v;
    }

    bool operator()(const OldClientsRecords& ar1, const OldClientsRecords& ar2) const {
        return ar1.getEmail() == ar2.getEmail();
    }
};
```

```
bool EmplAvailable::operator<(const EmplAvailable &eA) const {
    if(Available == eA.getSlot())
        return (e1->getUsername() > eA.getEmployee()->getUsername());
    else
    {
        DateAvail th(Available); DateAvail dA(eA.getSlot());

        return th < dA;
    }
}
```

Algoritmos relevantes



- Algoritmo que procura na fila de prioridade se existem dois slot seguidos vazios para serviços de duas horas.

```
if(task.getDuration() > 60) //Quando é mais do que um slot (2 - 120 min)
{
    found = false;
    h.addTime( minutes: 60); //Para ser o proximo slot
    if(eA.getEmployee()->slotExists(h))
    {
        twoSlots = true;
        //Se existir eliminar também da pq de Petwalker
        while(!EmplAvail.empty())
        {
            //Check if it is the day & hours we are looking for
            if((EmplAvail.top().getSlot() == h) && (EmplAvail.top().getEmployee()->getUsername() == eA.getEmployee()->getUsername()))
            {
                  eA = EmplAvail.top();
                EmplAvail.pop(); //tirar e não adicionar a Aux
                found = true;
                break; //break para ser apenas o primeiro nesse dia
            }
            else
            {
                Aux.push( x: EmplAvail.top());
                EmplAvail.pop();
            }
        }
```

Estrutura de Ficheiros

Gravação:

- Criada uma variável ofstream onde o ficheiro irá ser aberto;
- Lançada uma exceção se não for encontrado o ficheiro;
- Conforme a informação pretendida gravar, é escrito para a variável referida como no exemplo;
- Algo feito de forma diferente nesta parte do projeto foi o nome do cão associado ao serviço e dois ficheiros novos (cães e clientes inativos).
- Após ser escrita toda a informação pretendida, na ultima linha do ficheiro é introduzido um 'End' e o ficheiro é fechado;

```
for (auto c: allClients) {
    for (auto d: c->getDogs()) {
        dogs << d->getName() << endl;
        dogs << d->getRace() << endl;
        dogs << d->getBirth().getDay() << " " << d->getBirth().getMonth() << " " << d->getBirth().getYear() << endl;
        dogs << d->getOwnerEmail() << endl;
        dogs << endl;
    }
```

Leitura:

- Criada uma variável ifstream onde o ficheiro irá ser aberto;
- Lançada uma exceção se não for encontrado o ficheiro;
- São usadas istringstream para ler a informação, coloca-la em várias variáveis e inseri-las no local correto;
- Caso 'End' seja lido, o ficheiro é fechado.

```
istringstream client_user(line);
client_user >> client_email;
getline( &: services_done,  &: line);
istringstream dog_name(line);
dog_name >> name_dog;
getline( &: services_done,  &: line);
istringstream employee_user(line);
employee_user >> employee_username;
getline( &: services_done,  &: line);
istringstream excl(line);
excl >> exclusive;
getline( &: services_done,  &: line);
istringstream task_nam(line);
task_nam >> task_name;
```

Ler e criar (completa)

Read:

Lê os dados conforme o utilizador pretende (por serviço agendado, serviço completo, funcionário, cliente, cães, todas as disponibilidades, ou apenas as de um funcionário específico e clientes antigos).

Conforme o escolhido, aparece uma opção de organizar a forma de como os dados vão ser vistos.

Create:

Cria funcionários, clientes, adiciona um cão novo a um determinado cliente ou agenda um serviço, pedindo ao utilizador os respetivos dados.

Procurar e atualizar (completa)

Search:

Procura na base de dados pretendida pelo utilizador um objeto específico:

- Serviços por data;
- Serviços de uma tarefa;
- Serviços de um funcionário específico;
- Serviços de um cliente específico;
- Todos os cães de um específico cliente.

Update:

Atualiza os serviços como completos.

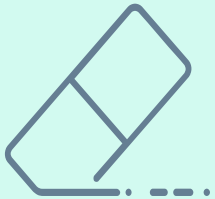
Também atualiza os preços das tarefas, a percentagem de lucro da empresa, o valor que os funcionários cobram e a quantidade de pontos que se ganha por preço que se paga.

Adiciona uma nova disponibilidade para um funcionário.

Atualiza o email do cliente.

Define um cliente como inativo (lança mensagem de erro caso o cliente posto tiver serviços agendados ou tiver realizado um serviço a menos de um mês.

Apagar (completa)



Delete:

Podem ser eliminados 3 objetos:

- Apagar um serviço agendado, sendo necessário introduzir os dados deste mesmo;
- Apagar um funcionário da empresa;
- Apagar um cliente da empresa;
- Remove uma disponibilidade para um funcionário.

Funcionalidade em Destaque

Ao eliminar um funcionário, os serviços agendados para este irão ser realocados para outro que esteja disponível.

Para além disso, para os serviços já realizados não desaparecerem do registo da empresa, o nome do funcionário eliminado passará a: NoMoreAEmployee.

Funcionalidade em Destaque


```
for(int i = 0; i < _scheduled_services.size();i++){
    if(_scheduled_services[i]->getEmployeeUsername() == "NoMoreAEmployee"){
        try{
            string substitute = checkDate( date: _scheduled_services[i]->getInitDate(),  exclusive: _scheduled_services[i]->isExclusive(),  task: _scheduled_services[i]->getTask());
            _scheduled_services[i]->setEmployeeUsername(substitute);
            cout << "\n  The service '" + _scheduled_services[i]->getTask().getTaskName() + "' at date '" + date +"' changed employee to '" + substitute + "'!";
        }
        catch (DateNotValid)
        {
            cout << "\n  The service '" + _scheduled_services[i]->getTask().getTaskName() + "' at date '" + date +"' was deleted!\n No available employees to substitute the deleted!\n";
            _scheduled_services.erase( position: _services_done.begin()+i);
            continue;
        }
        catch (EmployeeDoesNotExist &e)
        {
            cout << "\n\t" << e.getInfo() << endl;
        }
    }
}
```

Principais dificuldades encontradas

Dificuldades:

- Deparamo-nos com alguns problemas em partilhar o código no Git da FEUP, principalmente quando mexíamos no código ao mesmo tempo.

Esforço:

- Cada um teve a sua tarefa e conforme o tempo definido por nós, esta estava realizada a tempo.
- Podendo haver dificuldades nas tarefas distribuídas, todos os elementos estavam disponíveis para ajudar.

PetWalker

Turma 6 - Grupo 2

Ana Matilde Guedes Perez da Silva Barra
Maria Beatriz Russo Lopes dos Santos
Ricardo Filipe da Costa Cabral Ferreira

Image @ https://dogtime.com/assets/uploads/2018/10/how-to-get-pet-sitter-job-3.jpg