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).

- Os **serviços** podem ser efetuados de forma coletiva ou exclusiva (mais caro) por **funcionários** profissionais (preço mais elevado) ou de horas vagas.
- Os **clientes** possuem categorias (‘Silver’, ‘Gold’, ‘Platinum’) que é atribuído ao cliente dependendo do número de serviços que já efetuou e por cada 10€ pagos, o cliente recebe 1 ponto. Estes atributos terão benefícios no pagamento.

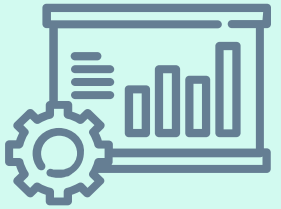
Descrição da Solução

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).

Estas classes possuem vários atributos que tornam possível a criação da empresa.

Algoritmos relevantes



- Atribuição de um serviço ao funcionário;
- Algoritmos que organizam os vetores com a informação da empresa;

```
sort(output.begin(), output.end(), [](Services *s1, Services *s2){
    if(s1->getServicePrice() == s2->getServicePrice())
        return (s1->getInitDate() < s2->getInitDate());
    else
        return (s1->getServicePrice() < s2->getServicePrice());
});
```

- Algoritmos que encontram num vetor um certo atributo e adicionam-no a outro.

```
for (int i = 0; i < _services_done.size();i++){
    if (_services_done.at(i)->getEmployeeUsername()==username){
        employee_done.push_back(_scheduled_services.at(i));
    }
}
```

Diagrama de Classes em UML

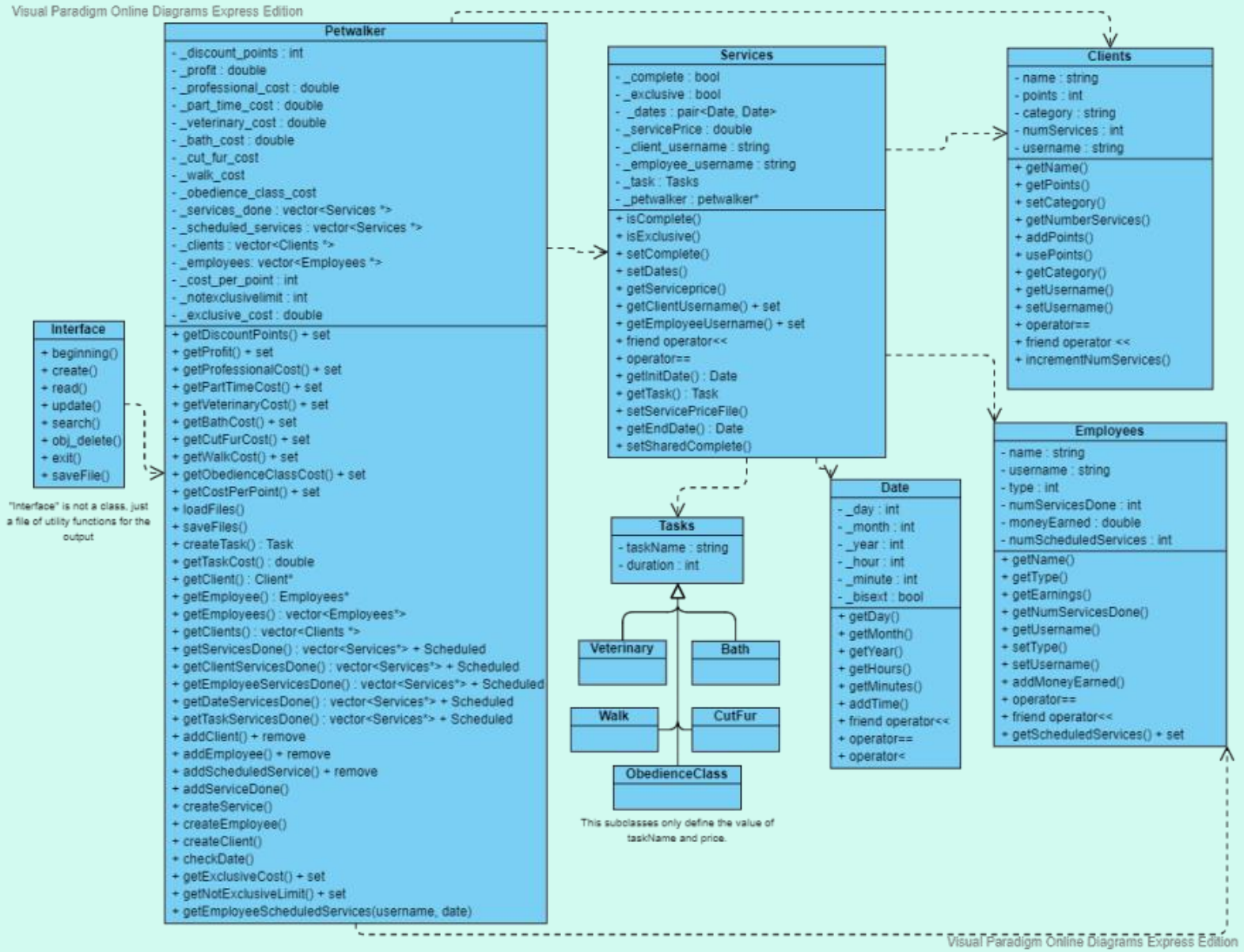


Diagrama de Classes em UML

Clients
- name : string - points : int - category : string - numServices : int - username : string
+ getName() + getPoints() + setCategory() + getNumberServices() + addPoints() + usePoints() + getCategory() + getUsername() + setUsername() + operator== + friend operator << + incrementNumServices()

Employees
- name : string - username : string - type : int - numServicesDone : int - moneyEarned : double - numScheduledServices : int
+ getName() + getType() + getEarnings() + getNumServicesDone() + getUsername() + setType() + setUsername() + addMoneyEarned() + operator== + friend operator<< + getScheduledServices() + set

Services
- _complete : bool - _exclusive : bool - _dates : pair<Date, Date> - _servicePrice : double - _client_username : string - _employee_username : string - _task : Tasks - _petwalker : petwalker*
+ isComplete() + isExclusive() + setComplete() + setDates() + getServiceprice() + getClientUsername() + set + getEmployeeUsername() + set + friend operator<< + operator== + getInitDate() : Date + getTask() : Task + setServicePriceFile() + getEndDate() : Date + setSharedComplete()

Diagrama de Classes em UML

Petwalker

- _discount_points : int
- _profit : double
- _professional_cost : double
- _part_time_cost : double
- _veterinary_cost : double
- _bath_cost : double
- _cut_fur_cost
- _walk_cost
- _obedience_class_cost
- _services_done : vector<Services *>
- _scheduled_services : vector<Services *>
- _clients : vector<Clients *>
- _employees: vector<Employees *>
- _cost_per_point : int
- _notexclusivelimit : int
- _exclusive_cost : double

+ getDiscountPoints() + set
+ getProfit() + set
+ getProfessionalCost() + set
+ getPartTimeCost() + set
+ getVeterinaryCost() + set
+ getBathCost() + set
+ getCutFurCost() + set
+ getWalkCost() + set
+ getObedienceClassCost() + set
+ getCostPerPoint() + set
+ loadFiles()
+ saveFiles()
+ createTask() : Task
+ getTaskCost() : double
+ getClient() : Client*
+ getEmployee() : Employees*
+ getEmployees() : vector<Employees*>
+ getClients() : vector<Clients *>
+ getServicesDone() : vector<Services*> + Scheduled
+ getClientServicesDone() : vector<Services*> + Scheduled
+ getEmployeeServicesDone() : vector<Services*> + Scheduled
+ getDateServicesDone() : vector<Services*> + Scheduled
+ getTaskServicesDone() : vector<Services*> + Scheduled
+ addClient() + remove
+ addEmployee() + remove
+ addScheduledService() + remove
+ addServiceDone()
+ createService()
+ createEmployee()
+ createClient()
+ checkDate()
+ getExclusiveCost() + set
+ getNotExclusiveLimit() + set
+ getEmployeeScheduledServices(username, date)

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;
- Após ser escrita toda a informação pretendida, na ultima linha do ficheiro é introduzido um 'End' e o ficheiro é fechado;

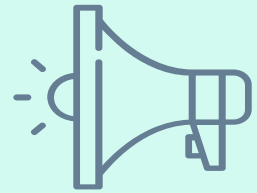
```
for (int i = 0; i < _clients.size();i++){
    clients << _clients[i]->getName() << endl;
    clients << _clients[i]->getUsername() << endl;
    clients << _clients[i]->getCategory() << endl;
    clients << _clients[i]->getPoints() << endl;
    clients << endl;
}
```

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.

```
if (line == "End")
    break;
istringstream nam(line);
nam >> first_name >> last_name;
name = first_name + " " + last_name;
getline( &: employees,  &: line);
istringstream user(line);
user >> username;
getline( &: employees, &: line);
istringstream typ(line);
typ >> type;
```

Tratamento de Exceções



- ClientDoesNotExist;
- EmployeeDoesNotExist;
- ServiceDoesNotExist;
- ClientAlreadyExists;
- EmployeeAlreadyExists;
- ServiceAlreadyExists;
- NoAvailableEmployee;
- InvalidServiceType;
- FileNotFound;
- EmptyVector;
- DateNotValid;
- InvalidInput.

```
class EmptyVector{
private:
    string info = "Vector is Empty!";
    int typeElemVector;
public:
    EmptyVector(){};
    EmptyVector(string inf) : info(inf){};
    EmptyVector(int type_elem) : typeElemVector(type_elem){
        switch (typeElemVector){
            case 0: //Elems of vector are services
                info = "Vector of Services is empty!";
                break;
            case 1: //Elems of vector are employees
                info = "Vector of Employees/Colaborators is empty!";
                break;
            case 2: //Elems of vector are clients
                info = "Vector of Clients is empty!";
                break;
            default:
                info = "Empty Vector.";
        }
    }
    string getInfo() {return info;};
```

```
class ClientDoesNotExist{
private:
    string username = "";
    string info = "\n Client " + username + " does not exist!\n";
public:
    ClientDoesNotExist(){};
    ClientDoesNotExist(string username):username(username){};
    string getInfo() {return info;};
};
```

```
class FileNotFound{
private:
    string info = "\n File " + namefile + " not found!\n";
    string namefile = "object.txt";
public:
    FileNotFound(){};
    FileNotFound(string namefile):namefile(namefile){};
    string getInfo() {return info;};
};
```

Ler e criar

Read:

Lê os dados conforme o utilizador pretende (por serviço agendado, serviço completo, funcionário, cliente).

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

Create:

Cria funcionários, clientes ou agenda um serviço, pedindo ao utilizador os respetivos dados.

Procurar e atualizar

Search:

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

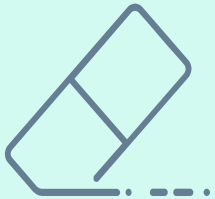
- Serviços por data;
- Serviços de uma tarefa;
- Serviços de um funcionário específico;
- Serviços de um cliente específico.

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.

Apagar



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;

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:

- Definir o serviço como completo sem recorrer a hora a que este é feito.
- Perde-se algum tempo a perceber o que outro elemento acrescentou ao código e a forma como o fez (cada um tem a sua forma de programar).

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