

# PROGRAMAÇÃO E DESENVOLVIMENTO DE SOFTWARE II

Trabalho Final

Daniel Quirino

Filipe Lavar

Lucas Mello

Pedro Vieira

# Introdução

## Objetivo

Usar de forma prática, conceitos aprendidos ao longo da disciplina como, encapsulamento, herança e polimorfismo.

## O projeto

O principal fim do projeto será a concepção de um sistema para a reciclagem de material.

Foram criados objetos primários, os quais foram bem encapsulados, que se definem: Pessoa, Resíduo, Ponto de coleta e Agendamento. Foi criada uma classe para organizar esses objetos, a classe cadastro, que possui funções derivadas para cada tipo de objeto. Como funcionalidade extra, a classe Ranking, imprime na tela, uma lista de maiores doadores e receptores.

O arquivo main.cpp mostra o funcionamento de cada classe assim como de cada função das classes.

# Implementação:

## Arquivo Resíduo.h

Classe responsável por definir em nível básico o que é um resíduo, material a ser reciclado, no código.

Ela contém a definição do material, uma string “\_material”. E também é possui uma string que explica características de armazenagem do resíduo, “\_descrição”. Possui uma variável bool “\_solido”, que explica de forma bem clara, se o material é sólido ou não.

A classe contém métodos que retornam cada uma das strings e bool, “*gets*”, e também métodos que conseguem mudar os valores das variáveis, “*sets*”.

```
7  class Residuo
8  {
9      public:
10         Residuo(bool _solido, string _material, string _descricao);
11
12         void setSolido(bool _solido);
13         void setMateria(bool _material);
14         void setDescricao(bool _descricao);
15
16         bool getSolido();
17         string getMaterial();
18         string getDescricao();
19
20     protected:
21
22     private:
23         bool solido;
24         string material;
25         string descricao;
26     };
```

## Arquivo Pessoa.h

Classe responsável por definir o que é uma pessoa para o algoritmo. E essa função é uma das principais do algoritmo, pois é justamente quem realiza ação de doar ou receber doação.

A função possui variáveis básicas para uma pessoa como: nome, endereço, data de nascimento, se é uma pessoa física, se é um doador ou não, CNPJ ou CPF.

Possui uma função para validar o CNPJ/CPF: “*void validarCpfCnpj(bool \_isPessoaFisica, string \_cpfCnpj)*”. Assim como possui uma função para verificar a data: “*void verificarData(string \_data)*”.

Existem funções também para retornar cada um dos valores das variáveis “*gets*”, assim como para modificá-las, “*sets*”.

```
9   class Pessoa
10  {
11      public:
12          Pessoa(string _nome, string _endereco, string _dataDeNascimento, bool _isPessoaFisica, bool _isDoador, double _cpfC
13              //virtual ~Pessoa();
14
15          void setNome(string _nome);
16          void setEndereco(string _endereco);
17          void setDataDeNascimento(string _dataDeNascimento);
18          void setIsPessoaFisica(bool _isPessoaFisica);
19          void setIsDoador(bool _isDoador);
20          void setCpfCnpj(int _cpfCnpj);
21          string getNome();
22          string getEndereco();
23          string getDataDeNascimento();
24          bool getIsPessoaFisica();
25          bool getIsDoador();
26          double getCpfCnpj();
27          void validarCpfCnpj(bool _isPessoaFisica, string _cpfCnpj);
28          void verificarData(string _data);
29
30      protected:
31
32      private:
33          string nome;
34          string endereco;
35          string dataDeNascimento;
36          bool isPessoaFisica; //1 para pessoa fisica
37          bool isDoador; // 1 para doador
38          double cpfCnpj;
39  };
```

## Arquivo PontoDeColeta.h

Responsável por definir uma localização, que será o ponto de encontro entre quem deseja doar resíduos e quem deseja receber.

Possui variáveis que armazenam dados sobre a localização: rua, bairro, número, complemento.

A classe possui funções que modificam e retornam as variáveis da classe, os “*sets*” e “*gets*”.

```

9   class PontoDeColeta
10  {
11      public:
12          PontoDeColeta( string _rua, string _bairro, int _numero, string _complemento);
13          string getRua();
14          string getBairro();
15          int getNumero();
16          string getComplemento();
17
18          void setRua(string _rua);
19          void setBairro(string _bairro);
20          void setNumero(int _numero);
21          void setComplemento(string _complemento);
22
23      protected:
24
25      private:
26          string rua;
27          string bairro;
28          int numero;
29          string complemento;
30
31  };
32

```

## Arquivo CadastroResíduo.h

Contém a classe CadastroResíduo, que consiste em um objeto responsável por organizar os resíduos(objeto) existentes no sistema.

Ela contém basicamente um vetor como atributo, e métodos que organizam esse vetor. Há funções: *create*, *update*, *deleted*, responsáveis por modificar os objetos resíduos dentro do vetor. Há a função *getDescrição*, que retorna uma *string* atributo do objeto contido em um ponto do vetor, que seria a descrição do material resíduo.

A função *listaDeResiduo()* imprime na tela todos os resíduos que constituem o vetor.

O construtor já cria cinco itens no vetor como default. Seriam esses os resíduos já pré-cadastrados no sistema.

```

10  class cadastroResiduo:Cadastro{
11      public:
12          cadastroResiduo();
13          ~cadastroResiduo();
14          void create(Residuo &r);
15          void update(Residuo &r, string _descricao);
16          void deleted(Residuo &r);
17          void listaDeResiduo();
18          string getDescricao(string _material);
19          vector<Residuo*> listaDeResiduos;
20
21      protected:
22      private:
23  };
24  #endif // CADATRORESIDOU_H_INCLUDED

```

## Arquivo CadastroPessoa.h

A classe CadastroPessoa é responsável por salvar as pessoas que se cadastraram no sistema. Ela possui essencialmente um vetor como atributo. E vários métodos que trabalham e organizam esse atributo.

Existem as funções: *create*, *update*, *deleted*, que criam, modificam e apagam algum item (pessoa) do vetor. E também possui a função *listarUsuarios()* que imprime os nomes dos usuários na tela.

```
13  class CadastroPessoa : Cadastro {
14
15  public:
16      CadastroPessoa(){}
17
18      void create(Pessoa &p);
19      void update(Pessoa &p, string _nome, string _endereco, string _dataDeNascimento, bool _isPessoaFisica, bool _isDoador, double
20      void deleted(Pessoa &p);
21      void listarUsuarios();
22      vector<Pessoa*> listaDeUsuarios;
23
24  protected:
25
26  private:
27
28  };
29
```

## Arquivo Agendamento. h

A classe agendamento é a responsável por conectar o doador ao receptor, assim como o resíduo de interesse, o local de coleta, e a data do encontro. Ela é o agendamento da doação de resíduo.

Essa classe possui como atributo, ponteiro para pessoa, receptor e doador, ponteiro para o resíduo e para o ponto de coleta, e uma string para a data. Possui métodos que retornam cada um desses atributos, além de funções “sets” para modificá-los.

Seu construtor recebe cada um desses valores para criar o objeto agendamento.

```
8
9  class Agendamento {
10      /** Agendamento */
11
12  public:
13      Agendamento(string _data, PontoDeColeta &_local, Pessoa &_doador, Pessoa &_receptor, string _residuos);
14
15      void setData(string _data);
16      void setLocal(PontoDeColeta &_local);
17      void setDoador(Pessoa &_doador);
18      void setReceptor(Pessoa &_receptor);
19      void setResiduos(string _residuos);
20
21      string getData();
22      PontoDeColeta getLocal();
23      Pessoa getDoador();
24      Pessoa getReceptor();
25      string getResiduos();
26
27  private:
28      string data;
29      PontoDeColeta *_local;
30      Pessoa *_doador;
31      Pessoa *_receptor;
32      string residuos;
33
34  };

```

## Arquivo CadastroAgendamento.h

A classe é responsável por organizar os objetos agendamentos em uma lista, vetor, e assim, poder gerenciá-los.

Possui como atributo um vetor de agendamentos, e métodos para modificá-lo. As funções *create*, *update*, *deleted* são as quais capazes de criar, modificar e retirar um agendamento na lista de agendamentos.

A função *listarAgendamentos()* imprime as datas dos agendamentos.

```
14  class CadastroAgendamento : Cadastro {
15
16  public:
17      CadastroAgendamento(){}
18
19      void create(Agendamento &p);
20      void update();
21      void deleted(Agendamento &p);
22      void listarAgendamentos();
23      vector<Agendamento*> listaDeAgendamentos;
24
25
26  protected:
27
28  private:
29
30  };
31
```

## Arquivo CadastroPontoDeColeta.h

Este arquivo define a classe CadastroPontoDeColeta, que é responsável por organizar todos os pontos de coleta adicionados no sistema.

Ela possui um vetor que guarda em ordem os objetos. Possui métodos *create*, *update*, *deleted*, que assim como em outras classes, conseguem criar, modificar e deletar objetos do vetor. Possui um método que imprime na tela as listas de pontos de coleta: *listarPontosDeColeta()*.

Seu construtor não possui parâmetros de entrada, e ele é vazio.

```
13  class CadastroPontoDeColeta : Cadastro {
14
15  public:
16
17      void create(PontoDeColeta &p);
18      void update(PontoDeColeta &p, string _rua, string _bairro, int _numero, string _complemento, string _id);
19      void deleted(PontoDeColeta &p);
20      void listarPontosDeColeta();
21      vector<PontoDeColeta*> listaDePontosDeColeta;
22
23  protected:
24
25  private:
26
27  };
```

## Arquivo Cadastro. h

A classe cadastro serve como classe abstrata, ou mais primitiva, para as demais classes cadastro. Ela define a idéia sobre o comportamento das classes cadastro, sendo ela que primeiro definiu as funções *create*, *update*, *daleted* e *listarobjetos*, que é utilizada em todas as derivadas. Ela é o formato básico para as CRUD utilizadas pelo trabalho.

```
7  class Cadastro
8  {
9  public:
10     Cadastro(){}
11     void create();
12     void update();
13     void deleted ();
14     void listarObjetos();
15
16     protected:
17
18     private:
19 };
20
```



## Arquivo Ranking.h

A classe ranking é a funcionalidade extra. Ela recebe no construtor um objeto da classe cadastroAgendamento, e através do vetor de agendamentos, extrai informações dos maiores doadores e receptores.

Ela recebe informações sobre doadores e receptores e as guarda em um mapa, em seqüência, copia e ordena em um vetor. E para isso, possui certas funções “*protected*” que são responsáveis por essa atividade. Elas são: *rankingDoar(CadastroAgendamento &cad)*, *rankingReceptor(CadastroAgendamento &cad)*, *insereDoador(Pessoa p)*, *insereReceptor(Pessoa p)* e *ordena()*. Essas funções são todas chamadas pelo construtor.

As funções que são publicas são: *maiorDoadorImprime()*, *maiorReceptorImprime()*, *rankingDoarLista()*, *rankingReceptorLista()* e *Ranking(CadastroAgendamento \*cad)*. Essas funções imprimem na tela o maior doador e o maior receptor, podendo também imprimir uma lista ordenada dos maiores doadores e receptores.

```
10  class Ranking{
11  public:
12      Ranking(CadastroAgendamento *cad);///construtor
13      void maiorDoadorImprime();
14      void maiorReceptorImprime();
15      void rankingDoarLista();
16      void rankingReceptorLista();
17  protected:
18      void insereDoador(Pessoa p);
19      void insereReceptor(Pessoa p);
20      void rankingDoar(CadastroAgendamento &cad);
21      void rankingReceptor(CadastroAgendamento &cad);
22      void ordena();
23
24      map<string,int> rankingDeDoar;
25      map<string,int> rankingDeReceptor;
26      vector<string> doador;
27      vector<int> _doador;
28      vector<string> receptor;
29      vector<int> _receptor;
30  private:
31  };
```

```
Maior doador:Daniel Pires com 2 doacoes agendadas
Maior receptor:Pedro Vieira com 1 recebimentos agendadas
Doador: #1 Daniel Pires com 2 doacoes agendadas
Receptor : #1 Pedro Vieira com 1 recebimento agendadas
Receptor : #2 Augusto da Silva com 1 recebimento agendadas
```

# Tratamento de Erros

## Entrada de dados inválidos para classe Pessoa

Caso tente entrar como data errada para o construtor, o programa informa que a data de nascimento esta inválida:

```
"C:\Users\lucas\OneDrive\UFMG\sexto semestre\pds2\PDS2-master\PDS2-n
Erro: Data de nascimento de Daniel Quirino invalida.

Pessoa *p0 = new Pessoa( Augusto da Silva , Rua Esplanada , 11/03/1903 , false, true , 010000000000111
Pessoa *p6 = new Pessoa("Daniel Quirino", "Rua João Fernandes", "30/02/1997", true, true , "13642091601");
Pessoa *p7 = new Pessoa("Filipe Lauar", "Avenida Miguel Perrela", "20/02/1996", true, false , "13494658678");
```

De forma semelhante, caso tente entrar com CPF ou CNPJ inválidos, o programa retorna erro na entrada desse dado.

```
"C:\Users\lucas\OneDrive\UFMG\sexto semestre\pds
Erro: Cpf de Filipe Lauar invalido.

Pessoa *p0 = new Pessoa( Daniel Quirino , Rua João Fernandes , 30/02/1997 , true, true , 13642091601 );
Pessoa *p7 = new Pessoa("Filipe Lauar", "Avenida Miguel Perrela", "20/02/1996", true, false , "13494658678");
```

Esse erro ocorre, pois toda vez que um CPF é inserido este, passa por uma função de teste: *validarCpfCnpj(bool \_isPessoaFisica, string \_cpfCnpj, string \_nome)*. Assim como, para a data, ocorreu um teste semelhante, em que a função responsável é: *verificarData(string \_data, string \_nome)*.

```
void Pessoa::validarCpfCnpj(bool _isPessoaFisica, string _cpfCnpj, string _nome)
{
    if(_isPessoaFisica)
    {
        string cpf = _cpfCnpj;
        string mensagemErroCpf = "Erro: Cpf de " + _nome + " invalido.";

        if (cpf.size() != 11)
            throw mensagemErroCpf;

        int i, numero, resultado, soma = 0, base = 10;
        int digitol, digito2;
        string str_numero, str_nono, str_decimo;

        for (i = 0; i <= 8; i++)
        {
            str_numero = cpf[i];
            numero = atoi(str_numero.c_str());
```

```


167 void Pessoa::verificarData(string _data, string _nome)
168 {
169     string data = _data;
170     string mensagemErroData = "Erro: Data de nascimento de " + _nome + " invalida.";
171
172     if(data.size() == 10 && ((data[2] == '/' && data[5] == '/') || (data[2] == '-' && data[5] == '-')))
173     {
174         string delimiter = "/";
175         string dia = data.substr(0, 2);
176         string mes = data.substr(3, 2);
177         string ano = data.substr(6, 4);
178
179         int _dia = atoi(dia.c_str());
180         int _mes = atoi(mes.c_str());
181         int _ano = atoi(ano.c_str());
182
183         if(_ano < 1900 || _ano > 2018 || _mes < 1 || _mes > 12 || _dia > 31 || _dia < 1)
184             throw mensagemErroData;
185         else if( dia > 28 && mes == 2)

```

## Entrada de dados inválidos para classe cadastroPessoa

A classe possui tratamento para erros de entrada.

Caso entre com datas erradas na função *update*, devido a classe pessoa, não aceitar datas invalidas, a classe cadastroPessoa, deste modo, também não aceita.

 "C:\Users\lucas\OneDrive\UFMG\sexto semestre\pds2\PDS2-master\PDS

Erro: Data de nascimento de Daniel Pires invalida.

```
cp->update(*p1, "Daniel Pires", "Rua João Fernandes 107", "10/05/2019", false);
```

Caso tente excluir alguém ou tentar dar update do cadastro que já não exista, o programa imprime para o usuário:

```

Usuario: Lucas Mello  nao Cadastrado
Usuario: Lucas Mello  nao Cadastrado
----- Lista atualizada de pessoas ca


```

```

cp->deleted(*p4);
cp->deleted(*p4);
cp->update(*p4, "Daniel Pires", "Rua João Fernandes 107", "10/05/1997", false);

```

A classe também não permite entrar com dois objetos pessoas iguais.

 "C:\Users\lucas\OneDrive\UFMG\sexto semestre\pds2\PDS

Usuario: Daniel Quirino ja Cadastrado  
----- Listar as pessoas cadastradas

```

CadastroPessoa *cp = new CadastroPessoa();
cp->create(*p1);
cp->create(*p1);
cp->create(*p3);

```

## Entrada de dados inválidos para classe cadastroResiduo

A classe não permite cadastrar dois objetos resíduos iguais. É impresso na tela: cout << "Residuo ja Cadastrado" << endl, para o usuário.

```
Residuo ja Cadastrado
----- Lista de residuo
// ~~~~~
cadastroResiduo *cadres= new cadastroResiduo();
cadres->create(*a);
cadres->create(*a);
cadres->create(*a);
```

Caso tente apagar um resíduo que não existe, o programa retorna erro ao usuário erro:

```
Residuo nao cadastrado
----- Lista de residuos

cadres->daleted(*e);
cadres->daleted(*e);
cadres->update(*b, "O vidro dev
```

## Entrada de dados inválidos para classe cadastroPontoDeColeta

Caso tente criar dois objetos iguais, o programa imprime erro para o usuário:

```
Ponto de coleta ja Cadastrado
----- Listar os pontos de coleta

cadPont->create(*pontCol2);
cadPont->create(*pontCol2);
cadPont->create(*pontCol4);
```

Assim como, o usuário tentar apagar um objeto que não exista na lista:

```
cadPont->deleted(*pontCol4);
cadPont->deleted(*pontCol4);
cadPont->update(*pontCol, "Rua Joao
```

## Entrada de dados inválidos para classe cadastroAgendamento

A classe, assim como as outras, não aceita itens iguais, e nem acessar a objetos inexistentes, ela retorna:

```

Agendamento: 1 ja Cadastrado
Agendamento: 2 nao Cadastrado
----- Lista de Agenda
00/00/2018 - Rua Joaquim Xavier

CadastroAgendamento *cadAgen = new CadastroAgendamento();
cadAgen->create(*ag1);
cadAgen->create(*ag1);
cadAgen->create(*ag3);

//CRUD com os agendamentos
cadAgen->deleted(*ag2);
-----

```

A classe também não aceita data erra, pois o objeto Agendamento não aceita, retorna o erro:

```

Erro: Data de 1 invalida.

```

## Entrada de dados inválidos para classe Agendamento

Caso entre com data errada, a classe retorna erro:

```

Erro: Data de 1 invalida.

```

```

//Criando Objetos de Agendamentos
Agendamento *ag1 = new Agendamento("18/19/2018", *pontCol, *p1, *p3, *a, "1");
Agendamento *ag2 = new Agendamento("00/00/2018", *pontCol, *p4, *p5, *b, "2");

```

# Conclusão

O trabalho mostrou a prática da matéria aprendida ao longo do semestre. Envolveu programação orientada a objetos, estrutura de dados, testes de softwares e boas práticas de desenvolvimento.

Além de exemplificar as dificuldades e vantagens de se programar em grupo. Já que, os envolvidos precisam seguir uma agenda de desenvolvimento para concluir o objetivo do projeto.

## Bibliografia

<http://www.cplusplus.com/reference/map/map/>

<http://www.cplusplus.com/reference/vector/vector/>

<https://pt.stackoverflow.com/questions/117906/como-criar-uma-classe-com-atributos-e-m%C3%A9todos-em-c>

<https://www.inf.pucrs.br/~pinho/PRGSWB/OO/oocpp.html>

[https://pt.wikipedia.org/wiki/Bubble\\_sort](https://pt.wikipedia.org/wiki/Bubble_sort)

[https://pt.wikibooks.org/wiki/Programar\\_em\\_C%2B%2B/Ponteiros](https://pt.wikibooks.org/wiki/Programar_em_C%2B%2B/Ponteiros)

