

# Classes e objetos - Herança

Roberto Rocha

# Programação Orientada a Objetos

# Encapsulamento

Em uma explicação literal do verbo, **encapsular** significa guardar algo em local fechado. Do ponto de vista da **orientação a objetos**, encapsular quer dizer ocultar todos os dados sobre um objeto, bem como detalhes da implementação de seus métodos.

Contudo, um objeto precisa ser utilizado por diferentes aplicações.

Essas aplicações acessam o objeto através de sua **interface pública**.

caso a implementação dos métodos mudar as mudanças ficarão restritas a essa classe.

Os métodos **getters e setters** representam bons exemplos dessa interface pública, uma vez que centralizam os acessos aos atributos privados, ou seja, as aplicações externas não visualizam os atributos em si, mas, sim, os métodos públicos liberados.

**Encapsular** é uma **excelente estratégia** para **manter ocultas** as **regras de negócio**, tornando-as visíveis apenas à classe responsável por elas. Contudo, ainda que as linguagens orientadas a objetos ofereçam recursos para garantir encapsulamento de objetos, seu emprego depende da compreensão de que a equipe de desenvolvedores possui do domínio da aplicação e da **maturidade** que possuem no uso do paradigma orientado a objetos.

# Encapsulamento em C++

Definição da classe Aluno:

```
#include <string>
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
using namespace std;
class Aluno
{
private:
    string nome;
    int matricula;
    string turma;
    string dataMatricula;
    void defineTurma();
public:
    Aluno();
    void setNome(string n);
    string getNome();
    string getTurma();
    string getDataMatricula();
};
```

Descrição dos métodos da classe Aluno:

```
void Aluno::defineTurma()
{
    turma=dataMatricula.substr(7,4);
}
Aluno::Aluno()
{
    dataMatricula= __DATE__;
    defineTurma();
}
void Aluno::setNome (string n)
{
    nome = n;
}
string Aluno::getNome()
{
    return nome;
}
string Aluno::getTurma()
{
    return turma;
}
string Aluno::getDataMatricula()
{
    return dataMatricula;
}
```

# Encapsulamento em C++

Definição da classe Aluno:

```
#include <string>
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
using namespace std;
class Aluno
{
private:
    string nome;
    int matricula;
    string turma;
    string dataMatricula;
    void defineTurma();
public:
    Aluno();
    void setNome(string n);
    string getNome();
    string getTurma();
    string getDataMatricula();
};
```

Exemplo de um  
programa principal:

```
int main()
{
    emcpp();
    emc();
    return 0;
}
```

Descrição dos métodos da classe Aluno:

Em C++

```
void emcpp()
{
    Aluno a;
    string strAux;
    cout << "Digite o nome do novo aluno:";
    fflush(stdin);
    cin>>strAux;
    a.setNome(strAux);
    cout << "Ficha cadastral do novo aluno:";
    cout << "\nnome:" << a.getNome();
    cout << "\nData da Matricula:" << a.getDataMatricula();
    cout << "\nTurma:" << a.getTurma() << "\n" << "\n" << "\n";
}
```

```
Digite o nome do novo aluno:florishbela
Ficha cadastral do novo aluno:
nome:florishbela
Data da Matricula: Jan 14 2016
Turma:2016
```

# Encapsulamento em C++

Definição da classe Aluno:

```
#include <string>
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
using namespace std;
class Aluno
{
private:
    string nome;
    int matricula;
    string turma;
    string dataMatricula;
    void defineTurma();
public:
    Aluno();
    void setNome(string n);
    string getNome();
    string getTurma();
    string getDataMatricula();
};
```

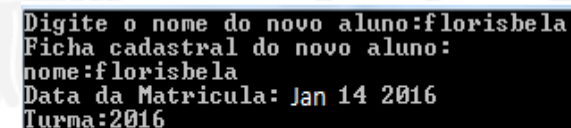
Exemplo de um programa principal:

```
int main()
{
    emcpp();
    emc();
    return 0;
}
```

Descrição dos métodos da classe Aluno:

## Em C

```
void emc()
{
    Aluno a;
    char strAux[50];
    printf("Digite o nome do novo aluno:");
    fflush(stdin);
    gets(strAux);
    a.setNome(strAux);
    printf("Ficha cadastral do novo aluno:");
    printf("\nnome:%s",a.getNome().c_str());
    printf("\nData da Matricula:%s",a.getDataMatricula().c_str());
    printf("\nTurma:%s\n\n",a.getTurma().c_str());
}
```



```
Digite o nome do novo aluno:florishela
Ficha cadastral do novo aluno:
nome:florishela
Data da Matricula: Jan 14 2016
Turma:2016
```

# Encapsulamento em C++

Definição da classe Aluno:

```
#include <string>
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
using namespace std;
class Aluno
{
private:
    string nome;
    int matricula;
    string turma;
    string dataMatricula;
    void defineTurma();
public:
    Aluno();
    void setNome(string n);
    string getNome();
    string getTurma();
    string getDataMatricula();
};

void Aluno::defineTurma()
{
    turma=dataMatricula.substr(7,4);
}
Aluno::Aluno()
{
    dataMatricula= __DATE__;
    defineTurma();
}
void Aluno::setNome (string n)
{
    nome = n;
}
string Aluno::getNome()
{
    return nome;
}
string Aluno::getTurma()
{
    return turma;
}
string Aluno::getDataMatricula()
{
    return dataMatricula;
}
```

Descrição dos métodos da classe Aluno:

Exemplo de um programa principal:

```
int main()
{
    emcpp();
    emc();
    return 0;
}
```

Em C++

```
void emcpp()
{
    Aluno a;
    string strAux;
    cout << "Digite o nome do novo aluno:";
    fflush(stdin);
    cin>>strAux;
    a.setNome(strAux);
    cout << "Ficha cadastral do novo aluno:";
    cout << "\nnome:" << a.getNome();
    cout << "\nData da Matricula:" << a.getDataMatricula();
    cout << "\nTurma:" << a.getTurma() << "\n" << "\n" << "\n";
}
```

Em C

```
void emc()
{
    Aluno a;
    char strAux[50];
    printf("Digite o nome do novo aluno:");
    fflush(stdin);
    gets(strAux);
    a.setNome(strAux);
    printf("Ficha cadastral do novo aluno:");
    printf("\nnome:%s",a.getNome().c_str());
    printf("\nData da Matricula:%s",a.getDataMatricula().c_str());
    printf("\nTurma:%s\n\n",a.getTurma().c_str());
}
```

```
Digite o nome do novo aluno:florishela
Ficha cadastral do novo aluno:
nome:florishela
Data da Matricula: Jan 14 2016
Turma:2016
```



# Encapsulamento em C++

Definição da classe Aluno:

```
#include <string>
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
using namespace std;
class Aluno
{
private:
    string nome;
    int matricula;
    string turma;
    string dataMatricula;
    void defineTurma();
public:
    Aluno();
    void setNome(string n);
    string getNome();
    string getTurma();
    string getDataMatricula();
};

void Aluno::defineTurma()
{
    turma=dataMatricula.substr(7,4);
}
Aluno::Aluno()
{
    dataMatricula= __DATE__;
    defineTurma();
}
void Aluno::setNome (string n)
{
    nome = n;
}
string Aluno::getNome()
{
    return nome;
}
string Aluno::getTurma()
{
    return turma;
}
string Aluno::getDataMatricula()
{
    return dataMatricula;
}
```

Descrição dos métodos da classe Aluno:

Exemplo de um programa principal:

```
int main()
{
    emcpp();
    emc();
    return 0;
}
```

Em C

```
void emc()
{
    Aluno a;
    char strAux[50];
    printf("Digite o nome do novo aluno:");
    fflush(stdin);
    gets(strAux);
    a.setNome(strAux);
    printf("Ficha cadastral do novo aluno:");
    printf("\nnome:%s",a.getNome().c_str());
    printf("\nData da Matricula:%s",a.getDataMatricula().c_str());
    printf("\nTurma:%s\n\n",a.getTurma().c_str());
}
```

Crie uma classe e instancie um objeto para a classe carro.

Atributos privados: - marca, ano de fabricação e placa

Públicos cor, valor, cpf do proprietário

Método IPVA: 5% do valor do carro - 10% por ano de uso.

Crie os métodos getters e setters



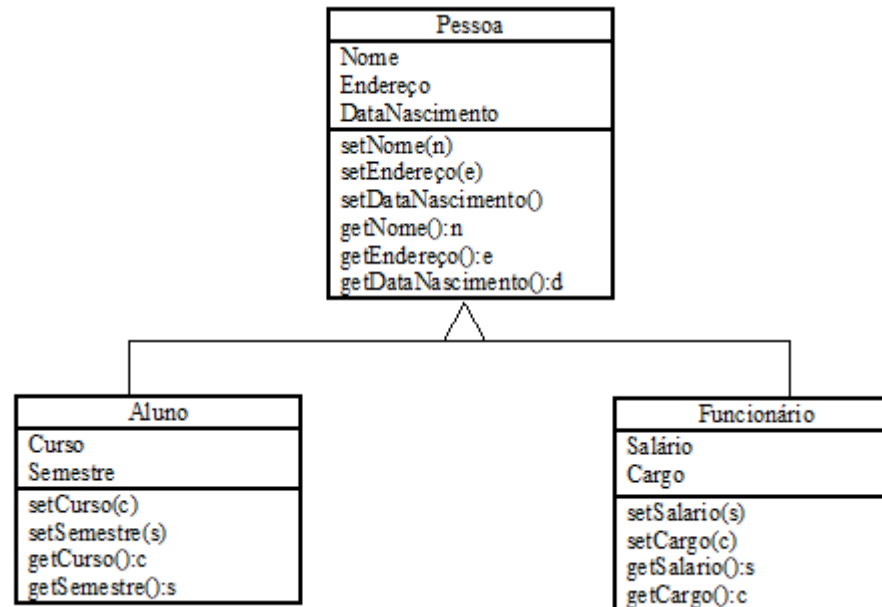
# Herança

Um dos grandes recursos proporcionados pela Orientação a Objetos é a **Herança**.

A **Herança** tenta organizar em classes, chamadas superclasses, todos os atributos e métodos comuns a vários tipos de objetos.

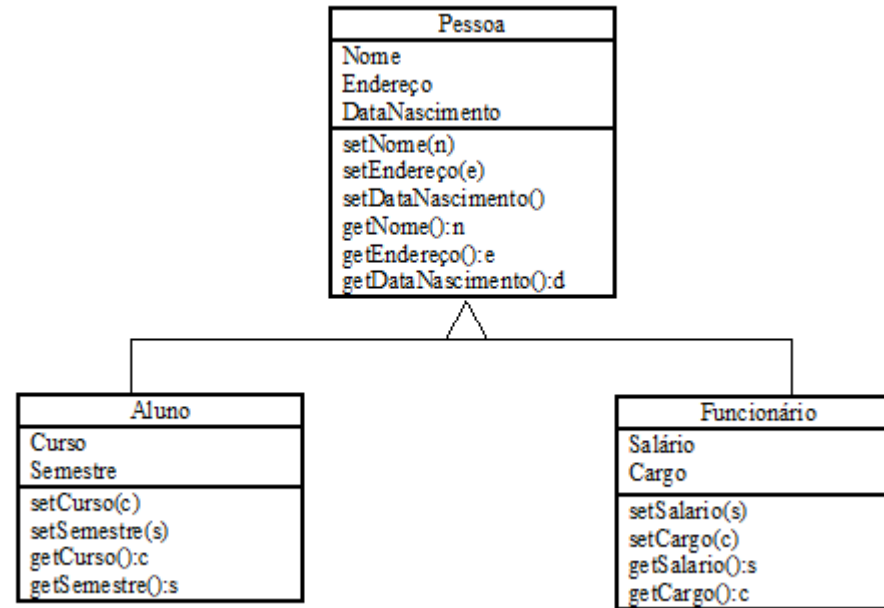
Caso alguns objetos possuam particularidades, estas deverão ser descritas em classes chamadas **subclasses**. Dizemos que **subclasses** estendem **superclasses**.

Herança permite implementar relacionamentos do tipo “é um”.



# Herança

Herança permite implementar relacionamentos do tipo “é um”.



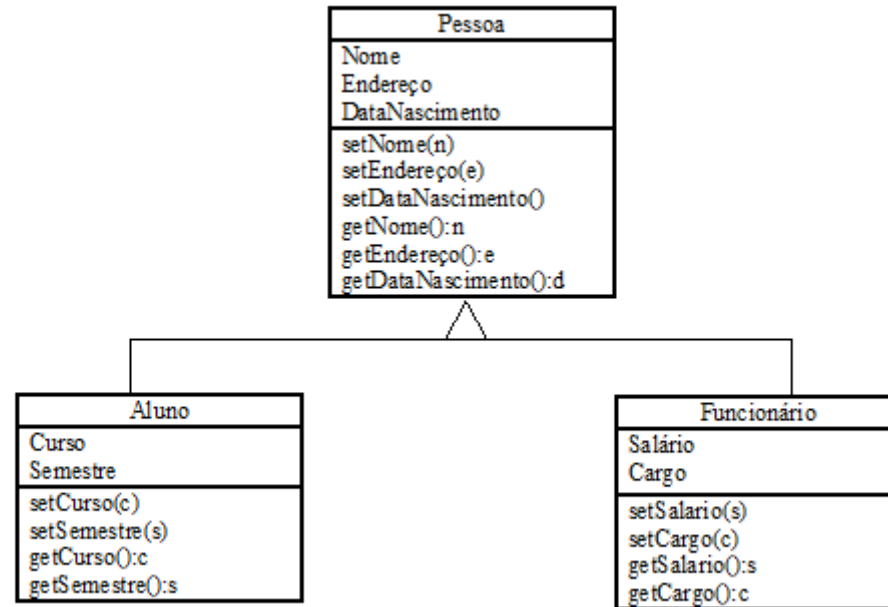
Temos três classes relacionadas por meio de **herança**. Cada caixa representa uma classe e o triângulo representa a ocorrência da **herança**. A base do triângulo está sempre voltada para as classes filhas.

A classe **Pessoa**, representa o conjunto de todas as pessoas, quer sejam **alunos** quer **funcionário**.

A classe **Pessoa** é, portanto, a **superclasse**.

# Herança

Herança permite implementar relacionamentos do tipo “é um”.



A **classe Aluno** representa as especificidades observadas apenas em alunos.

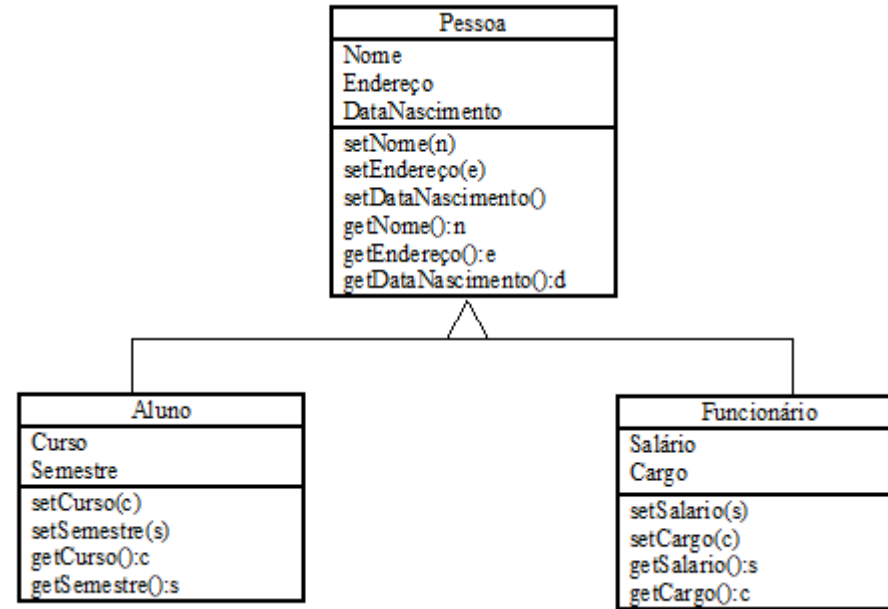
A **classe Aluno** é uma subclasse de **Pessoa**.

Isso quer dizer que, quando a **classe Aluno** for instanciada, o objeto criado conterá tudo o que está descrito na **classe Aluno** e herdará tudo o que estiver descrito na **classe Pessoa**.

Conclui-se então que um aluno terá Nome, Endereço, DataNascimento, Curso e Semestre.

# Herança

Herança permite implementar relacionamentos do tipo “é um”.



A **classe Funcionário** representa as especificidades observadas apenas em funcionários.

A **classe Funcionário** é uma subclasse de **Pessoa**.

Isso quer dizer que, quando a **classe Funcionário** for instanciada, o objeto criado conterá tudo o que está descrito na **classe Funcionário** e herdará tudo o que estiver descrito na **classe Pessoa**.

Conclui-se então que um funcionário terá Nome, Endereço, DataNascimento, Salario e Cargo.

**As classes Aluno e Funcionário são classes irmãs e não compartilham entre si qualquer informação.**

# Herança em C++

Exemplo:

## SuperClasse Pessoa

```
#include <iostream>
#include <string>
using namespace std;

class Pessoa
{
private:
    string nome;
    string endereco;
    string dataNascimento;
public:
    Pessoa();
    string getNome();
    void setNome(string n);
    string getEndereco();
    void setEndereco(string e);
    string getDataNascimento();
    void setDataNascimento(string d);
};
```

## Métodos da classe Pessoa

```
Pessoa::Pessoa()
{
    cout << "\nExecutando construtor da classe Pessoa";
}

string Pessoa::getNome() {
    return nome;
}

void Pessoa::setNome( string n) {
    nome=n;
}

string Pessoa::getEndereco() {
    return endereco;
}

void Pessoa::setEndereco(string e) {
    endereco = e;
}

string Pessoa::getDataNascimento() {
    return dataNascimento;
}

void Pessoa::setDataNascimento( string d) {
    dataNascimento = d;
}
```

# Herança em C++

## SubClasse Aluno

```
class Aluno: public Pessoa
{
private:
    string curso;
    string semestre;
public:
    Aluno();
    string getCurso();
    void setCurso( string c);
    string getSemestre();
    void setSemestre ( string s);
};
```

## Métodos da classe Aluno

```
Aluno::Aluno() {
    cout << "\nExecutando construtor da classe Aluno";
}
string Aluno::getCurso() {
    return curso;
}
void Aluno::setCurso( string c) {
    curso =c;
}
string Aluno::getSemestre() {
    return semestre;
}
void Aluno::setSemestre( string s) {
    semestre=s;
}
```

# Herança em C++

## SubClasse Funcionário

```
class Funcionario: public Pessoa
{

private:
    float salario;
    string cargo;
public:
    Funcionario();
    float getSalario();
    void setSalario( float s);
    string getCargo();
    void setCargo( string c);
};
```

## Métodos da classe Funcionário

```
Funcionario::Funcionario() {
    cout << "\nExecutando construtor da classe Funcionario";
}
float Funcionario::getSalario() {
    return salario;
}
void Funcionario::setSalario( float s) {
    salario =s;
}
string Funcionario::getCargo() {
    return cargo;
}
void Funcionario::setCargo( string c) {
    cargo = c;
}
```



# Herança em C++

## SuperClasse Pessoa

```
class Pessoa
{
private:
    string nome;
    string endereco;
    string dataNascimento;
public:
    Pessoa();
    string getNome();
    void setNome(string n);
    string getEndereco();
    void setEndereco(string e);
    string getDataNascimento();
    void setDataNascimento(string d);
};
```

## SubClasse Aluno

```
class Aluno: public Pessoa
{
private:
    string curso;
    string semestre;
public:
    Aluno();
    string getCurso();
    void setCurso( string c);
    string getSemestre();
    void setSemestre ( string s);
};
```

## SubClasse Funcionário

```
class Funcionario: public Pessoa
{
private:
    float salario;
    string cargo;
public:
    Funcionario();
    float getSalario();
    void setSalario( float s);
    string getCargo();
    void setCargo( string c);
};
```

# Herança em C++

## SuperClasse Pessoa

## SubClasse Aluno      SubClasse Funcionário

### Programa principal

```
int main()
{
    cout << "Criando um objeto da classe Pessoa";
    Pessoa p;
    p.setNome("Joao");
    p.setEndereco("Rua Joao e Maria , 123");
    p.setDataNascimento("01/01/2001");

    cout << "\n\nCriando um objeto da classe Aluno";
    Aluno a;
    a.setNome("Maria");
    a.setEndereco("Rua Maria e Joao , 321");
    a.setDataNascimento("02/02/2002");
    a.setCurso("Engenharia de Software");
    a.setSemestre("2/2016");

    cout << "\n\nCriando um objeto da classe Funcionario";
    Funcionario f;
    f.setNome("Jose");
    f.setEndereco("Rua Maria Joao e Jose, 43321");
    f.setDataNascimento("03/03/1980");
    f.setCargo("Presidente");
    f.setSalario(50000);
}
```

// impressao dos dados dos registros

```
cout << "\n\nDados cadastrados do objeto da classe Pessoa";
cout << "\nNome:" << p.getNome();
cout << "\nEndereco:" << p.getEndereco();
cout << "\nData de Nascimento:" << p.getDataNascimento();

cout << "\n\nDados cadastrados do objeto da classe Aluno";
cout << "\nNome:" << a.getNome();
cout << "\nEndereco:" << a.getEndereco();
cout << "\nData de Nascimento:" << a.getDataNascimento();
cout << "\nCurso:" << a.getCurso();
cout << "\nSemestre:" << a.getSemestre();

cout << "\n\nDados cadastrados do objeto da classe Funcionario";
cout << "\nNome:" << f.getNome();
cout << "\nEndereco:" << f.getEndereco();
cout << "\nData de Nascimento:" << f.getDataNascimento();
cout << "\nCargo:" << f.getCargo();
cout << "\nSalario:" << f.getSalario();

return 0;
}
```

```
Criando um objeto da classe Pessoa
Executando construtor da classe Pessoa

Criando um objeto da classe Aluno
Executando construtor da classe Pessoa
Executando construtor da classe Aluno

Criando um objeto da classe Funcionario
Executando construtor da classe Pessoa
Executando construtor da classe Funcionario

Dados cadastrados do objeto da classe Pessoa
Nome:Joao
Endereco:Rua Joao e Maria , 123
Data de Nascimento:01/01/2001

Dados cadastrados do objeto da classe Aluno
Nome:Maria
Endereco:Rua Maria e Joao , 321
Data de Nascimento:02/02/2002
Curso:Engenharia de Software
Semestre:2/2016

Dados cadastrados do objeto da classe Funcionario
Nome:Jose
Endereco:Rua Maria Joao e Jose, 43321
Data de Nascimento:03/03/1980
Cargo:Presidente
Salario:50000
Process returned 0 (0x0)   execution time : 1.952 s
Press any key to continue.
```

# Herança em C++

## SubClasse Funcionário

## SuperClasse Pessoa

```
class Pessoa
{
private:
    string nome;
    string endereco;
    string dataNascimento;
public:
    Pessoa();
    string getName();
    void setName(string n);
    string getEndereco();
    void setEndereco(string e);
    string getDataNascimento();
    void setDataNascimento(string d);
};
```

## SubClasse Aluno

```
20
21 class Aluno: public Pessoa
22 {
23 private:
24     string curso;
25     string semestre;
26 public:
27     Aluno();
28     string getCurso();
29     void setCurso(string c);
30     string getSemestre();
31     void setSemestre(string s);
32 };
```

```
33
34 class Funcionario:public Pessoa
35 {
36
37 private:
38     float salario;
39     string cargo;
40 public:
41     Funcionario();
42     float getSalario();
43     void setSalario(float s);
44     string getCargo();
45     void setCargo(string c);
46 };
```

## Programa principal

```
122 // aplicação principal
123 int main()
124 {
125     cout << "Criando um objeto da classe Pessoa";
126     Pessoa p;
127     p.setName("Joao");
128     p.setEndereco("Rua Joao e Maria , 123");
129     p.setDataNascimento("01/01/2001");
130
131     cout << "\n\nCriando um objeto da classe Aluno";
132     Aluno a;
133     a.setName("Maria");
134     a.setEndereco("Rua Maria e Joao , 321");
135     a.setDataNascimento("02/02/2002");
136     a.setCurso("Engenharia de Software");
137     a.setSemestre("2/2016");
138
139     cout << "\n\nCriando um objeto da classe Funcionario";
140     Funcionario f;
141     f.setName("Jose");
142     f.setEndereco("Rua Maria Joao e Jose, 43321");
143     f.setDataNascimento("03/03/1980");
144     f.setCargo("Presidente");
145     f.setSalario(50000);
146 }
```

```
147 // impressao dos dados dos registros
148
149 cout << "\n\nDados cadastrados do objeto da classe Pessoa";
150 cout << "\nNome:" << p.getName();
151 cout << "\nEndereco:" << p.getEndereco();
152 cout << "\nData de Nascimento:" << p.getDataNascimento();
153
154 cout << "\n\nDados cadastrados do objeto da classe Aluno";
155 cout << "\nNome:" << a.getName();
156 cout << "\nEndereco:" << a.getEndereco();
157 cout << "\nData de Nascimento:" << a.getDataNascimento();
158 cout << "\nCurso:" << a.getCurso();
159 cout << "\nSemestre:" << a.getSemestre();
160
161 cout << "\n\nDados cadastrados do objeto da classe Funcionario";
162 cout << "\nNome:" << f.getName();
163 cout << "\nEndereco:" << f.getEndereco();
164 cout << "\nData de Nascimento:" << f.getDataNascimento();
165 cout << "\nCargo:" << f.getCargo();
166 cout << "\nSalario:" << f.getSalario();
167
168 return 0;
169 }
```

## Exercício:

Crie uma SuperClasse denominada Veiculo

Com os atributos private: ano de fabricação, placa e km atual

Agora crie duas subclasses: uma automóvel com os atributos de veiculo e mais os atributos próprios: numero de portas, km ultima revisao a outra subclasse será caminhão contendo os atributos próprios: nr de pneus, capacidade e valor frete por km.

Crie todos os métodos getters e setters



**PUC Minas**  
**Virtual**