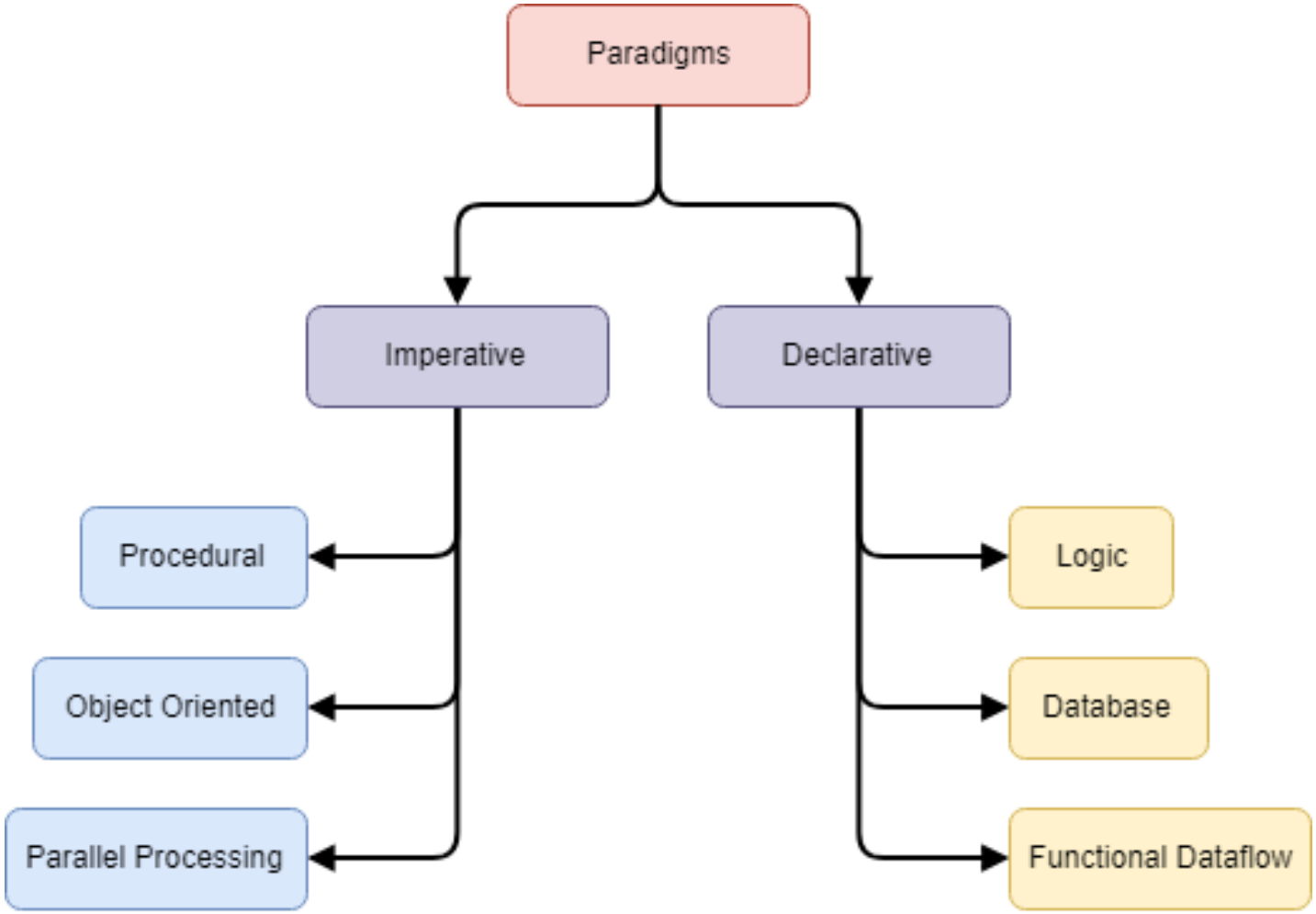
Orientação a Objetos
Introdução

Prof. Renato Coral Sampaio

Paradigmas de Programação

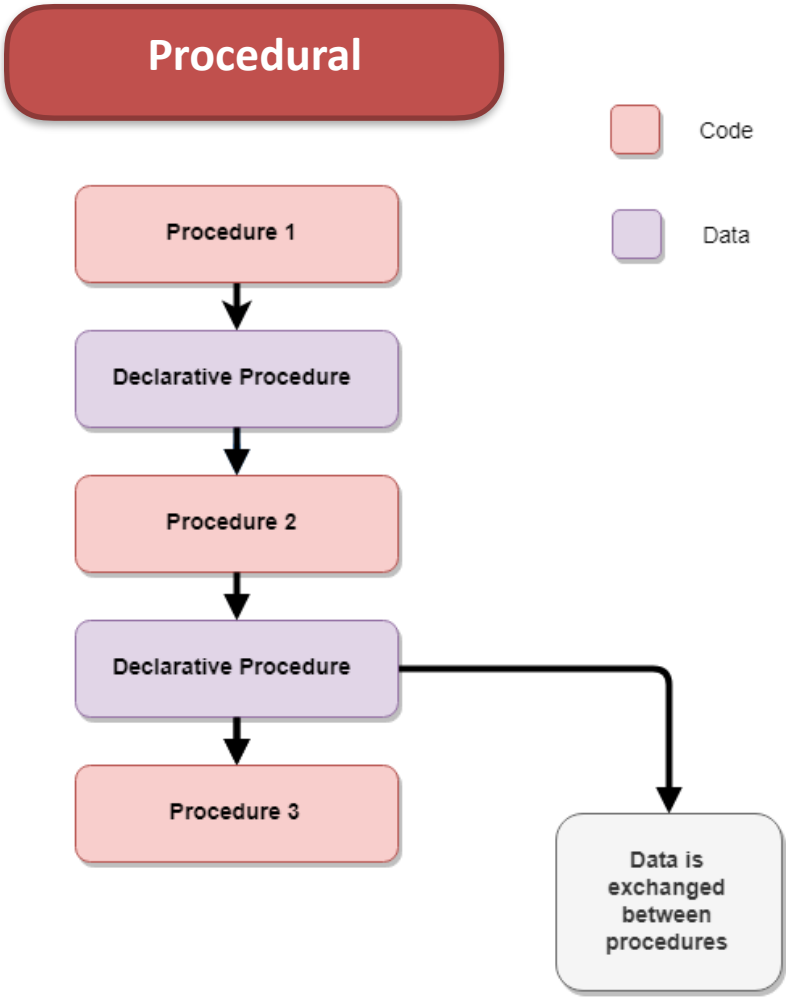
- Sem estrutura
  - Programas pequenos e muitas vezes com acesso a variáveis globais
  - O mesmo código deve ser repetido para executar um procedimento mais de uma vez.
- Procedural
  - O programa fica mais estruturado e pode ser visto como uma sequência de *procedures.* Um programa com várias partes.
- Modular
  - Funcionalidades são agrupadas em módulos.
  - Cada módulo é único.
- Orientada a Objetos
- Funcional

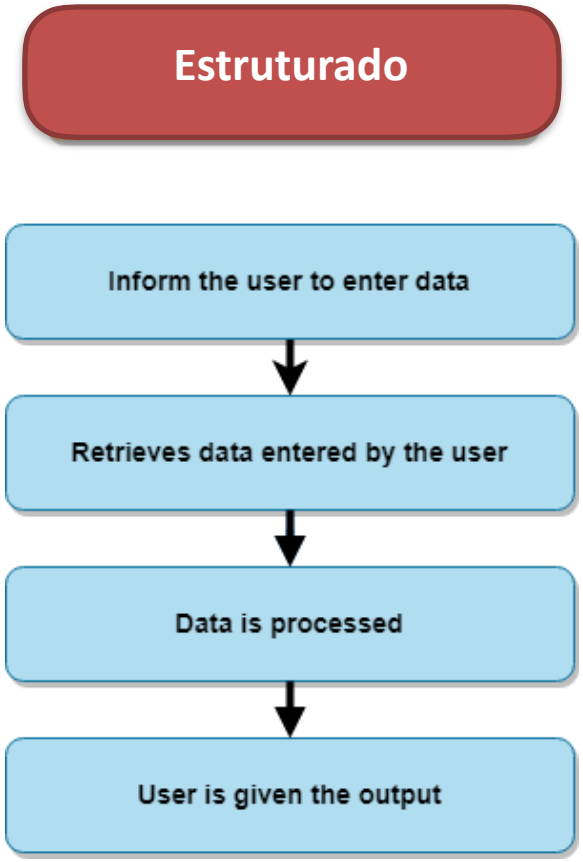
Paradigmas de Programação



Ref.: https://ctechf.com/programming-part-1-introduction/

Paradigmas de Programação

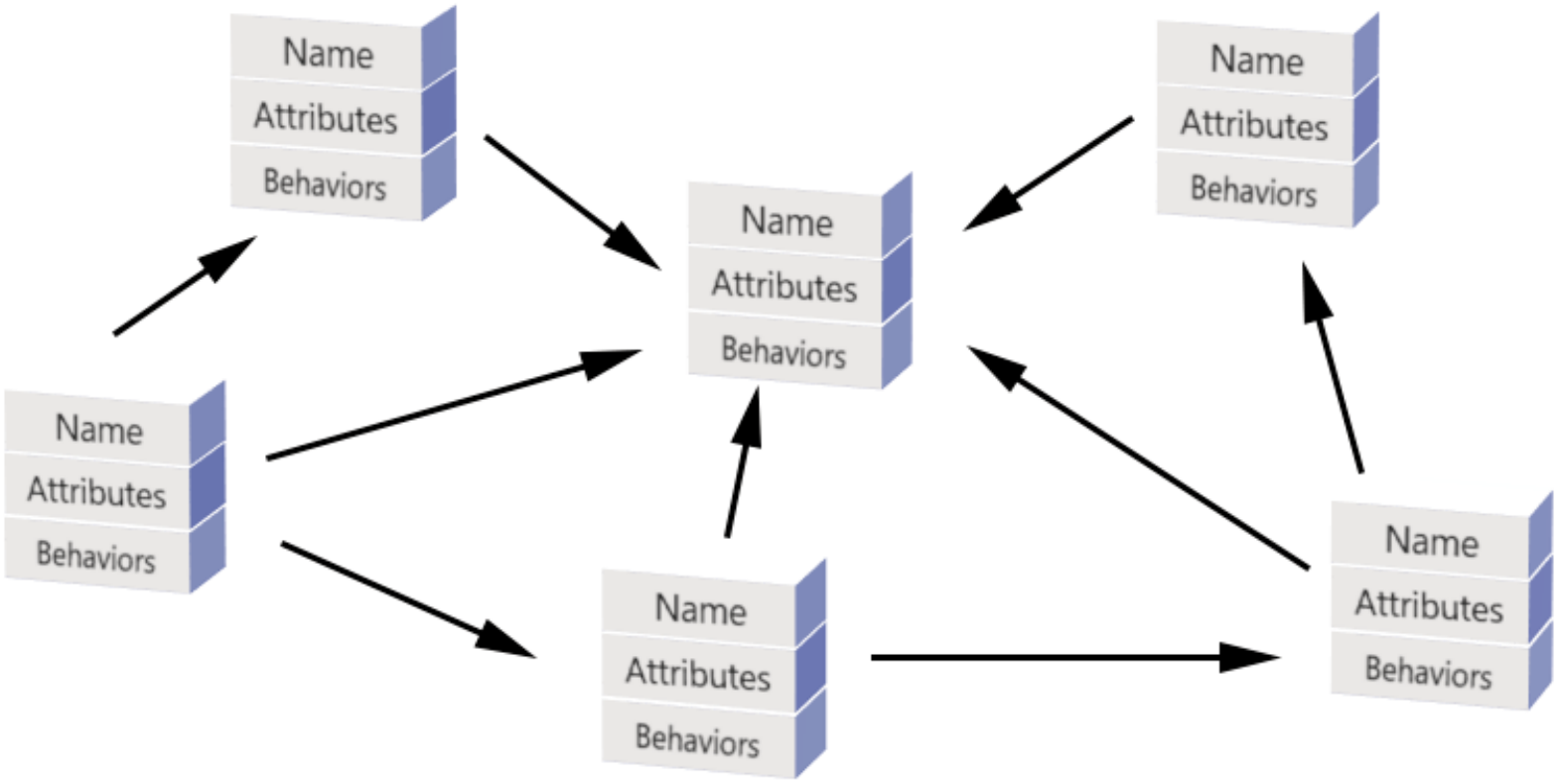




Ref.: https://ctechf.com/programming-part-1-introduction/

Paradigmas de Programação

Orientado a Objetos



Ref.: https://ctechf.com/programming-part-1-introduction/

Conceitos Básicos

- Programação Procedural (Código contínuo)
- Criação de Objetos para melhorar a organização
- Conceito para aumentar a aderência com a realidade
- Orientação a Objetos é um Paradigma de programação.

Orientação a Objetos

- O que é?
  - Paradigma de Programação baseado em Objetos
  - Baseada no princípio da reutilização de componentes

Orientação a Objetos

- Para que serve?
  - Organização
  - Colaboração
  - Reaproveitamento / Reusabilidade
  - Flexibilidade
  - Melhoria da Manutenabilidade do código

Linguagens Orientadas a Objeto

- C++
- C#
- Java
- JavaScript
- Perl
- PHP
- Python
- Objective-C
- Ruby
- VB.NET
- Swift

Fundamentos de OO

- Abstração
  - Objeto
  - Classe
- Encapsulamento
- Herança
- Polimorfismo

Abstração

- Extrair a essência de objetos da realidade
- Capacidade de compreender o contexto ao qual cada objeto pertence e definir as características essenciais do mesmo para este determinado contexto
- Exemplo:
  - Carro (venda) vs. Carro (fábrica)

Objeto

- O que é um objeto?
  - Algo que existe e tem identidade própria
  - Um objeto pode conter outro objeto
  - Cada objeto possui características ou **atributos** que descrevem seu estado
  - A maioria dos objetos possui vários atributos
  - Além disso, um objeto tem **comportamento**

Objeto

- **Exemplos**:
  - ***Objetos concretos***: Copo, Caneta, Mochila, Carro
  - ***Objetos abstratos***: conta de banco, conexão de rede.
- Uma maneira de identificar se algo pode ser caracterizado como objeto é ver se ele é descrito por um substantivo.

Classe

- Usada para criar objetos
- Descreve o que o objeto será, é um modelo do objeto, uma planta.
- Composta por
  - Nome
  - Atributos (Propriedades, Dados)
  - Métodos (Operações, Comportamento)

Atributos

- Atributos são as características de um objeto. É a estrutura de dados que vai representar a classe.
- Ex: Classe Pessoa - Atributos: nome, endereço, telefone, CPF,...; Classe Carro - Atributos: nome, marca, ano, cor, ...; Classe Livro - Atributos: autor, editora, ano.
- O valor de cada atributo identifica o objeto e informa seu estado.

Métodos

- Definem os comportamentos dos objetos.
- São normalmente são públicos, sendo assim os meios de interação da entre classes.
- Tipos:
  - **Construtores**: responsáveis pela alocação de memória e inicialização de dados, sendo sempre chamados automaticamente na declaração um novo objeto
  - **Destrutores**: chamados quando o objeto é destruído. Liberam a memória, fecham arquivos, conexões, etc.
  - Métodos de acessores (**Get** e **Set**)
  - Outros

Exemplo: Classe Pessoa em C++

Pessoa
string nome int idade string telefone
Pessoa() setNome() getNome() setIdade() getIdade() setTelefone() getTelefone()

pessoa.hpp

```
#include <iostream>
#include <string>

class Pessoa {
private:
        string  nome;
        string  idade;
        string  telefone;
public:
        Pessoa();
        Pessoa(string nome, string idade, string telefone);
        string  getIdade();
        void setIdade(string idade);
        string getNome();
        void setNome(string nome);
        string getTelefone();
        void setTelefone(string telefone);
};
```

Exemplo: Classe Pessoa em C++

pessoa.cpp

```
#include "pessoa.hpp"
#include <string>

using namespace std;

Pessoa::Pessoa(){
        nome = "";
        idade = "";
        telefone = "";
}
Pessoa::Pessoa(string nome, string
idade, string telefone){
    setNome(nome);
    setIdade(idade);
    setTelefone(telefone);
}
string  Pessoa::getNome() {
        return nome;
}
void Pessoa::setNome(string nome) {
        this->nome = nome;
}
```

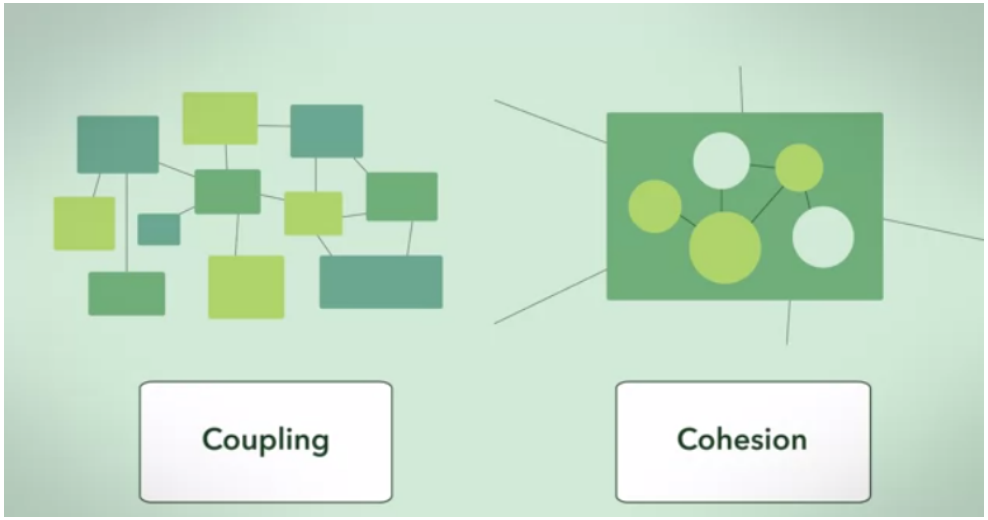
```
string  Pessoa::getIdade() {
        return idade;
}

void Pessoa::setIdade(string idade)
{
        this->idade = idade;
}

string  Pessoa::getTelefone() {
        return telefone;
}

void Pessoa::setTelefone(string
telefone) {
        this->telefone = telefone;
}
```

Encapsulamento

- Controla a visibilidade/acesso à Atributos e Métodos
  - **Público**: podem ser acessados por qualquer entidade no programa.
  - **Privado**: tem acesso restrito aos membros da própria classe e as classes amigas (*friends*).
  - **Protegido**: tem acesso restrito aos membros da própria classe, as classes filhas (herança) e as classes amigas (*friends*).

Encapsulamento

- Diagrama de Acoplamento e Desacoplamento de Classes
  - Acoplamento e Coesão
  - ...



Ref. : https://www.coursera.org/lecture/object-oriented-design/1-3-1-coupling-and-cohesion-q8wGt

Encapsulamento

- Diagrama de Acoplamento e Desacoplamento de Classes
  - Acoplamento e Coesão

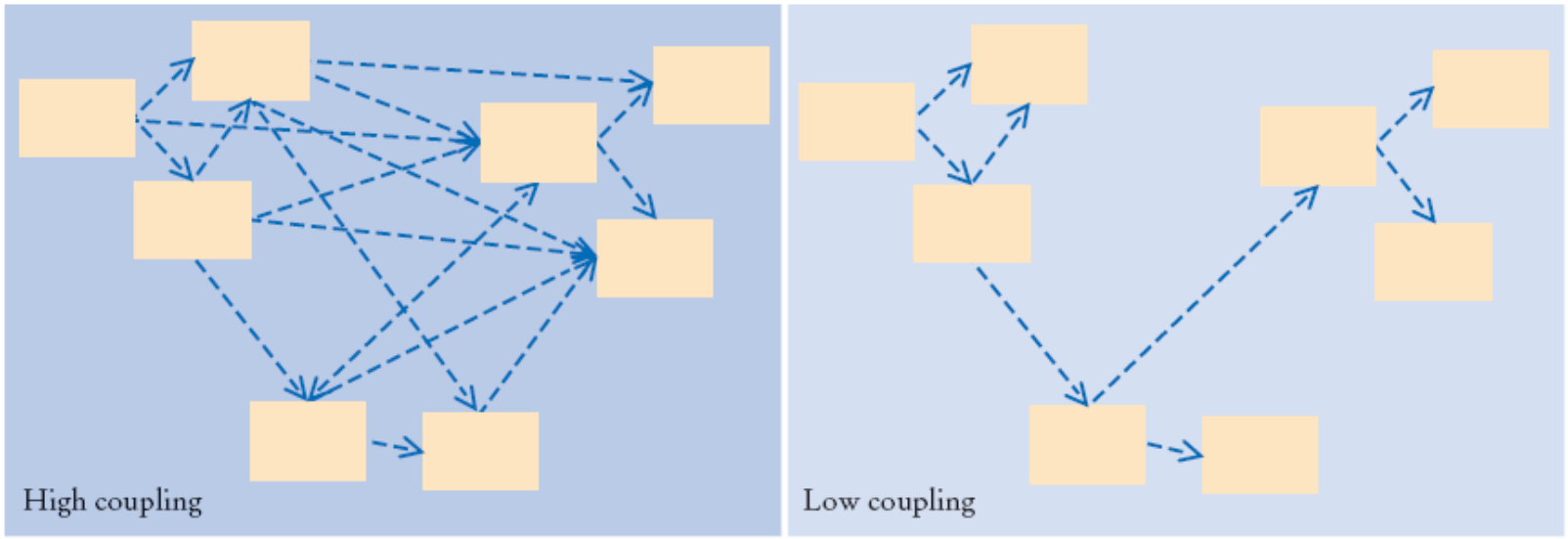


Figure 2 High and Low Coupling Between Classes

Ref. : https://www.iro.umontreal.ca/~pift1025/bigjava/Ch09/ch09.html

Encapsulamento

- Atenção!
  - Ao retornar um ponteiro para um atributo ou método privado ou protegido de sua classe você está praticamente o tornando público.
  - Qualquer entidade do programa que tenha o endereço de memória de um atributo ou método, na prática tem acesso direto ao mesmo.
  - Este pode ser considerado uma brecha na implementação do C++.

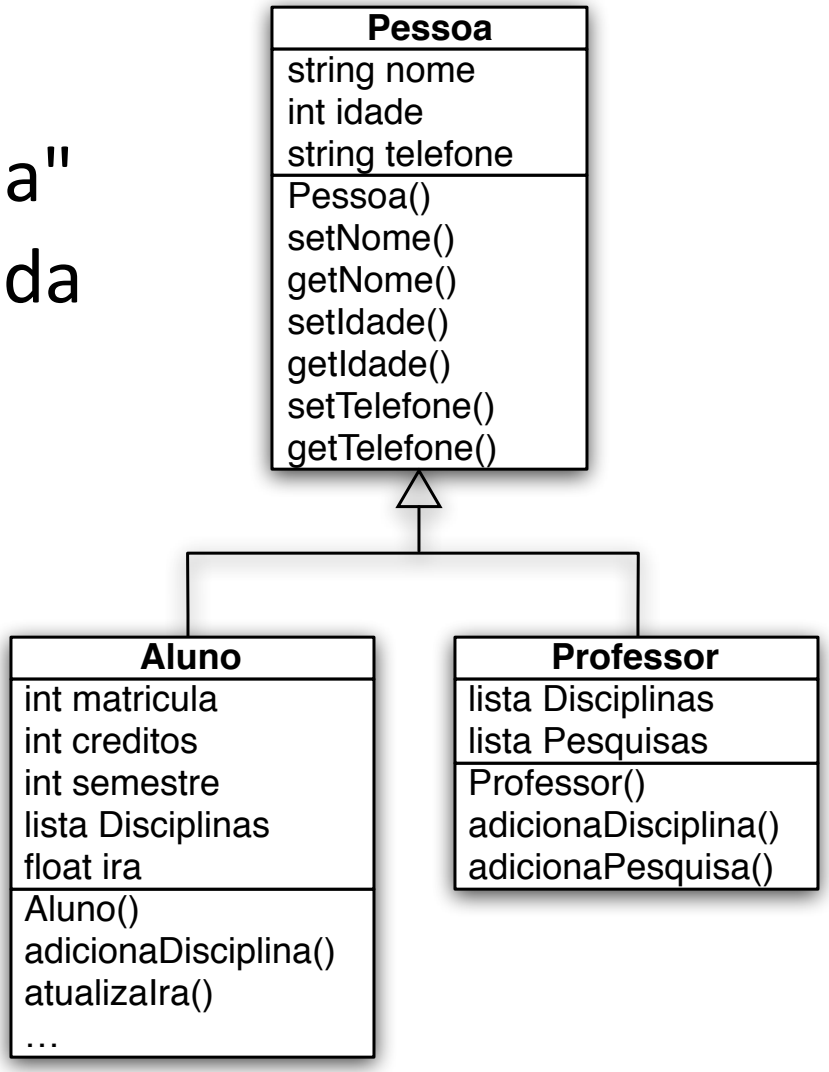
Ref. https://www.quora.com/How-do-we-access-private-data-members-of-a-class-outside-the-program-in-C++-language

Herança

- Princípio que permite classes compartilharem atributos e métodos.
- Viabiliza o reaproveitamento e especialização de classes.
- **Tipo**:
  - Herança Simples
  - Herança Múltipla
  - Herança em múltiplos níveis

Herança

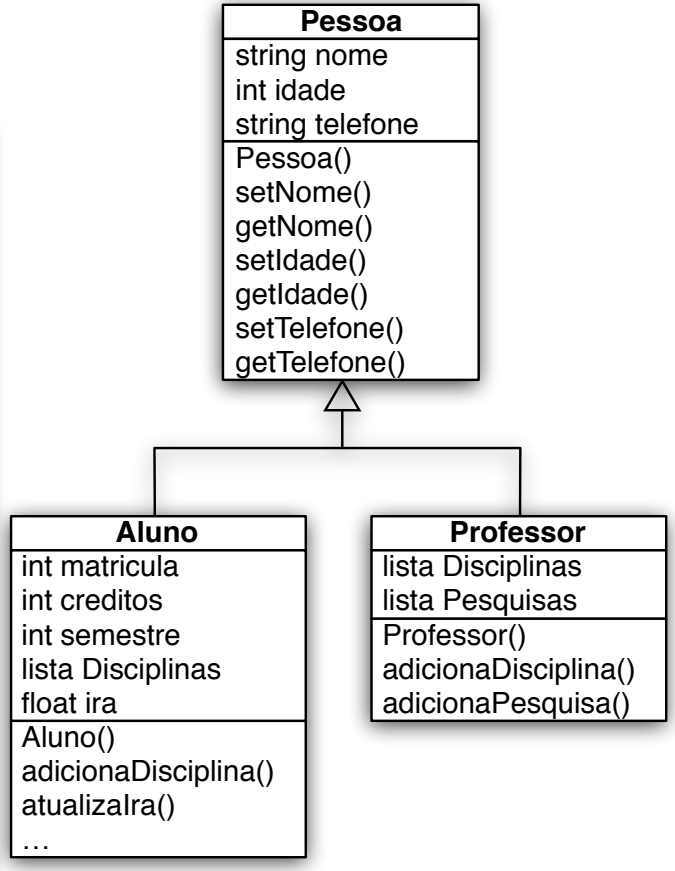
- A Classe "filha" ou “derivada" herda atributos e métodos da classe "pai"



Herança

aluno.hpp

```
#include "pessoa.hpp"

class Aluno: public Pessoa
{
    private:
     int matricula;
     int quantidade_de_creditos;
     int semestre;
     float ira;
    public:
     void Aluno();
     void Aluno(string nome, string idade, string
telefone, int matricula);
     void setMatricula(int matricula);
     int getMatricula();
     void setQuantidadeCreditos(int creditos);
     int getQuantidadeCreditos();
     void setSemestre(int semestre);
     int getSemestre();
     string getSemestreString();
     void setIra(float ira);
     float getIra();
};
```



Polimorfismo

- É um meio de prover uma interface única para entidades de tipos diferentes.
- Tipos:
  - **Sobrecarga**: quando métodos ou operadores de mesmo nome em uma classe recebem parâmetros diferentes.
  - **Sobrescrita**: quando métodos de classes derivadas possuem mesma assinatura do método da superclasse (classe “pai”) porém funcionam de maneiras distintas.

Polimorfismo - Exemplos

- Sobrecarga de métodos:

Métodos com mesmo nome ...

```
int calculaArea(int base, int altura);

float calculaArea(float base, float altura);
```

```
void Aluno();

void Aluno(string nome, string idade, int matricula);
```

com parâmetros distintos

Polimorfismo - Exemplos

- Sobrescrita de métodos:

```
class FormaGeometrica {
    ...
    float calculaArea(){
        return base * altura;
    }

};

class Triangulo : public FormaGeometrica {
    ...
    float calculaArea(){
        return base * altura / 2;
    }
};
```

Métodos com mesma assinatura ...

mas com implementações distintas.

Polimorfismo - Exemplos

- Sobrecarga de Operadores:

```
int x, y, z;
  z = x + y;
float f1, f2, f3;
  f3 = f1 + f2;
string s1, s2, s3;
  s3 = s1 + s2;
complexo c1, c2, c3 => no formato (a + bi)
  c3 = c1 + c2;
```

Polimorfismo - Exemplos

- Sobrecarga de Operadores - sintaxe

```
class nomeDaClasse
{
    ... .. ...
    public:
       tipoDeRetorno operator símbolo (argumentos)
       {
           ... .. ...
       }
    ... .. ...
};
```