Univiçosa – Centro Universitário de Viçosa
Cursos: Análise e Desenvolvimento de Sistemas e Engenharia de Computação
Disciplina: ADS101/ECO131 – Programação de Computadores 2

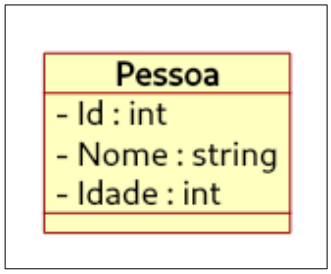
Aula 04 – Criando um métodos na classe

1) Introdução

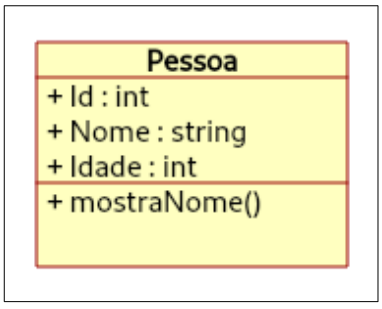
Como visto anteriormente um objeto é a representação material de algo do mundo real. Além de seus atributos também temos os métodos ou ações, estes são aquilo que o objeto pode fazer dentro do escopo da classe.

2) Adicionando os métodos na classe criada no UML

Vimos que uma classe é a representação de um objeto. Este objeto possui atributos que lhe são inerentes. Usamos a classe Pessoa para representar o objeto pessoa e 3 atributos detalhados na imagem abaixo.



Usando o UML para visualmente criarmos a estrutura da classe vamos adicionar um método. Criamos o método “mostraNome”. Este método será inserido no código da classe posteriormente. Observe a imagem abaixo.



3) Criando o código do método dentro da classe

Inicialmente temos o código usado para criar o primeiro exemplo do C++ com orientação a objetos.

Observe o código abaixo.

```
#include <iostream>
#include <string>
#include <conio.h>

using namespace std;

class Pessoa{

   public:
      int Id;
      string Nome;
      int Idade;
};

int main(){

   Pessoa pessoa;

   pessoa.Id = 1;
   pessoa.Nome = "José";
   pessoa.Idade = 20;


   cout << "Id: " << pessoa.Id << endl;
   cout << "Nome: " << pessoa.Nome << endl;
   cout << "Idade: " << pessoa.Idade << endl;

   return 0;
}
```

Vamos inserir o método criado no UML dentro da classe Pessoa. O método é basicamente uma função e segue a mesma sintaxe desta. O código ficará como no quadro abaixo.

```
class Pessoa{

   public:
      int Id;
      string Nome;
      int Idade;

      void getNome(){

         cout << Nome;
```

```
    }
};
```

Explicando o código.

Ao criar o “void getNome()” todos os atributos da classe ficam disponibilizados para uso dentro do método. Se colocarmos uma quebra de linha e inserir a “Idade” e o “Id” dentro do método eles seriam exibidos quando este for chamado na execução do programa como no quadro abaixo.

```
void getNome(){

        cout << "Id: " << Id << endl;
        cout << "Nome: " << Nome << endl;
        cout << "Idade: " << Idade << endl;
    }
```

4) Adicionando segurança no acesso aos atributos

Um programa certamente terá informações que são importantes e não devem ser acessadas por outros módulos do próprio programa ou por programas externos. Na orientação a objetos temos a implementação do recurso de encapsulamento. Este encapsulamento pode ser de 3 formas:

- Public - a visibilidade é aberta a todos. Isso significa que qualquer classe pode acessar e usar esse atributo;
- Private – somente a classe que criou o atributo pode usá-la;
- Protected - a visibilidade é apenas para a classe derivada, quando há uma herança. Isso significa que qualquer classe derivada pode acessar e usar este atributo.

Vamos alterar o programa e implementar o encapsulamento nos atributos da classe Pessoa. Observe o quadro abaixo.

```
class Pessoa{

  private:
    int Id;
    string Nome;
    int Idade;


    public:
    void setNome (string _nome);
    string getNome();
};
```

Trocamos o “public:” por “private:” na criação dos atributos, impedindo que outras classes e programas acesse-os diretamente. Depois criamos um encapsulamento do tipo “public” para as próximas duas funções. A primeira função é:

```
void setNome (string _nome);
```

Esta função usa o recurso de “setter”, ou em português “setar” um valor. Como parâmetro passamos o tipo “string” e criamos uma variável temporária “_nome”.

Logo abaixo fazemos uma alteração no método “getNome()”. Ao invés de passarmos a ação que ela fará, criamos um protótipo do método que será posteriormente definido logo abaixo.

Logo abaixo do fechamento da classe passamos para a definição e implementação de cada um dos métodos criados. Observe o quadro abaixo.

```
void Pessoa::setNome(string _nome){
      Nome = _nome;
   }

string Pessoa::getNome(){
      return Nome;
   }
```

Explicando o código.

Temos na primeira linha o uso de “::” entre o nome da classe e o método. Este uso é característico nas linguagens que trabalham com OOP. Chama-se primeiro a classe e, com os “::” definimos qual método será usado. Como já definimos o atributo “Nome”, chamamos e atribuímos a ele a variável provisória que foi “setada” no método.

O método “getNome()” deste código é a implementação do que foi prototipado através da linha

```
string getNome();
```

Neste método é chamado o comando para exibir na saída padrão o nome definido.

A forma como foram definidos os dados de cada um dos atributos não podem ser mais usados por causa do encapsulamento definido na classe. O código listado abaixo não pode ser usado.

```
pessoa.Id = 1;
pessoa.Nome = "José";
pessoa.Idade = 20;
```

Estas linhas podem ser comentadas ou até mesmo apagadas.

A forma para definir que “Nome” receberá o valor “José” será feito da seguinte forma. No lugar onde estavam as 3 linhas deve-se escrever.

```
Pessoa.setNome(“José”);
```

O código completo do programa está no quadro abaixo.

```
#include <iostream>
#include <string>
#include <conio.h>

using namespace std;

class Pessoa{

   private:
      int Id;
      string Nome;
      int Idade;

      public:
      void setNome (string _nome);
      string getNome();
};


   void Pessoa::setNome(string _nome){
      Nome = _nome;
   }

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

int main(){

   Pessoa pessoa;

   pessoa.setNome("José");
   cout << "Nome: " << pessoa.getNome();


   return 0;
}
```

Com esta nova estrutura usamos os conceitos de OOP.

5) Considerações da aula

Nesta aula inserimos o conceito de método, encapsulamento e o acesso aos atributos utilizando as técnicas da orientação a objetos. Nas próximas aulas conheceremos mais recursos da orientação a objetos aplicados ao C++.

Bons estudos.