ALGORITMOS E ESTRUTURAS DE DADOS I

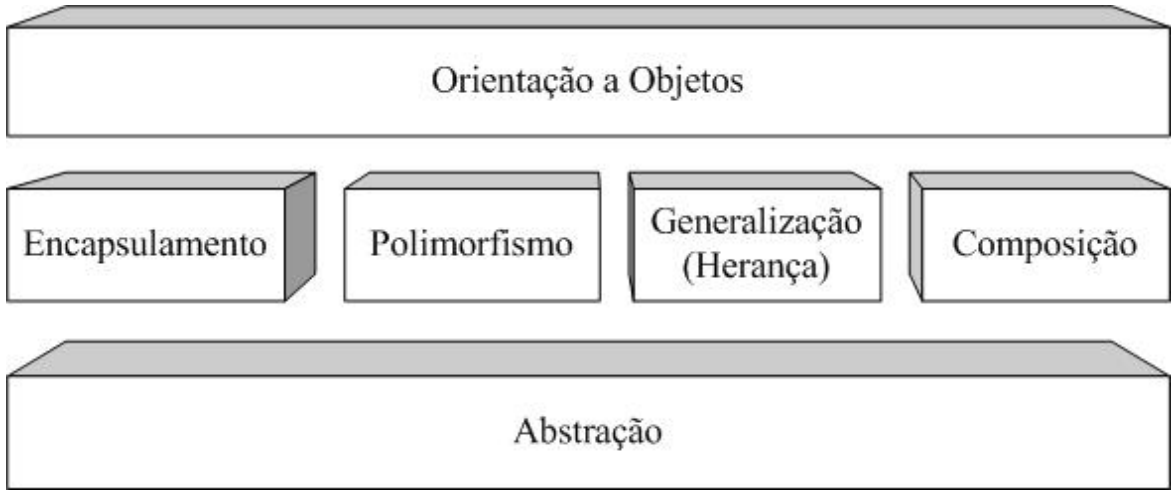
UNIDADE 7

INTRODUÇÃO À ORIENTAÇÃO A OBJETOS

PROF. NAÍSSES ZÓIA LIMA

Princípios da Orientação a Objetos

- A programação orientada a objetos (POO) baseia-se em **três princípios básicos**:
  - Encapsulamento
  - Composição / Herança
  - Polimorfismo



Encapsulamento

- Um dos pontos essenciais de POO é o de esconder as estrutura de dados dentro de certas entidades (objetos),
  - aos quais são associados métodos (funções) que manipulam essas estruturas de dados.
- Na orientação a objetos, esconder os detalhes de implementação de uma classe é um conceito conhecido como **encapsulamento**.
- Como os detalhes de implementação da classe estão escondidos, todo o acesso deve ser feito através de seus métodos públicos. Não permitimos aos outros saber **COMO** a classe faz o trabalho dela, mostrando apenas **O QUÊ** ela faz.

Encapsulamento

- Trabalhamos com **struct** até o momento. Structs definem apenas atributos, mas não métodos.
- Para definir métodos, usamos funções externas à struct que recebem uma variável do tipo da struct.
- Em C++, introduziu-se a funcionalidade de classe e todo o suporte a orientação à objetos.
- Com classes, podemos definir atributos bem como métodos, ou seja, objetos possuem características (atributos) e comportamentos (métodos).
- Objetos são variáveis cujos tipos são classes. Ao criarmos uma variável do tipo de uma classe, estamos instanciando um objeto.

Encapsulamento

- Qualificadores de Acesso:

+ public : um método definido como public pode ser acessado por qualquer classe de qualquer projeto

- private : é mais restritivo, somente a classe onde ele foi definido é que pode acessá-lo, nenhuma outra tem permissão, nem mesmo classes que herdam da classe onde o método foi definido

 # protected : somente as classes que herdam da classe que contem o método protegido tem permissão para acessá-lo e as classes que estão no mesmo pacote.

Hello World C++

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Ola mundo!" << endl;
    return 0;
}
```

Classes

- Com classes, podemos definir atributos bem como métodos, ou seja, objetos possuem características (atributos) e comportamentos (métodos).
- Objetos são variáveis cujos tipos são classes. Ao criarmos uma variável do tipo de uma classe, estamos instanciando um objeto.
- Para criar um objeto, usamos uma função específica para criação chamada de **construtor**.
- Todo objeto possui um ponteiro para si mesmo denominado **this**.
- A classe pode ter métodos com mesmo nome, porém com lista de parâmetros distintos: **sobrecarga** de métodos.

Classes

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

class Quadrado {
private:
   int lado;

public:
   Quadrado(int l): lado(l) {}

   int getLado() {
      return lado;
   }

   int perimetro() {
      return 4*lado;
   }

   int area() {
      return lado * lado;
   }

   void print() {
      cout << "[Quadrado, lado=" << this->lado << "]" <<
endl;
   }
};
```

Classes

```
int main()
{
   //Constroi quadrado de lado 10
   Quadrado q(10);

   //Imprime o lado do quadrado
   cout << "lado do quadrado = " << q.getLado()
<< endl;

   //Imprime o perimetro do quadrado
   cout << "perimetro do quadrado = " <<
q.perimetro() << endl;

   //Imprime o area do quadrado
   cout << "area do quadrado = " << q.area() <<
endl;

   //Imprime o quadrado
   q.print();

   return 0;
}
```

```
lado do quadrado = 10
perimetro do quadrado = 40
area do quadrado = 100
[Quadrado, lado=10]

Process returned 0 (0x0)   execution time : 0.056 s
Press any key to continue.
```

Classes

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

class Quadrado {
private:
   int lado;

public:
   Quadrado(): lado(0) {}
   Quadrado(int l): lado(l) {}

   void setLado(int lado) {
      this->lado = lado;
   }

   int getLado() {
      return lado;
   }

   int perimetro() {
      return 4*lado;
   }

   int area() {
      return lado * lado;
   }

   void print() {
      cout << "[Quadrado, lado=" << this->lado << "]" <<
endl;
   }
};
```

Classes

```
int main()
{
   //Constroi quadrado
   Quadrado q;

   //Configura o lado do quadrado como 20
   q.setLado(20);

   //Imprime o lado do quadrado
   cout << "lado do quadrado = " << q.getLado() <<
endl;

   //Imprime o perimetro do quadrado
   cout << "perimetro do quadrado = " << q.perimetro()
<< endl;

   //Imprime o area do quadrado
   cout << "area do quadrado = " << q.area() << endl;

   //Imprime o quadrado
   q.print();

   return 0;
}
```

```
lado do quadrado = 20
perimetro do quadrado = 80
area do quadrado = 400
[Quadrado, lado=20]

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

Construtores e Destrutores

- A criação de objetos é feita por meio de **construtores**. São funções com o mesmo nome da classe e sem retorno, que inicializam o objeto e seus atributos.
- Construtor default:
  - é um construtor que pode ser chamado sem argumentos;
  - se uma classe não tem nenhum construtor, é criado automaticamente pelo compilador um construtor default;
  - porém, a criação de qualquer construtor pelo programador, faz com que este construtor default não seja mais criado automaticamente!

Construtores e Destrutores

- Destrutor é função complementar às funções construtoras de uma classe. Sempre que o escopo de um objeto encerra-se, esta função é chamada.
- Cada classe pode ter somente um destrutor que jamais recebe parâmetros. O destrutor também não tem nenhum tipo de retorno.

```
class Y {
  public:
    ~Y();
};
```

Alocação Dinâmica

- Em C++, alocação dinâmica de memória é feita usando o operador new e new[].

```
pointer = new type
pointer = new type [number_of_elements]
```

```
int * pInt = new int; //1 inteiro
int * pInts = new int[5]; //5 inteiros
```

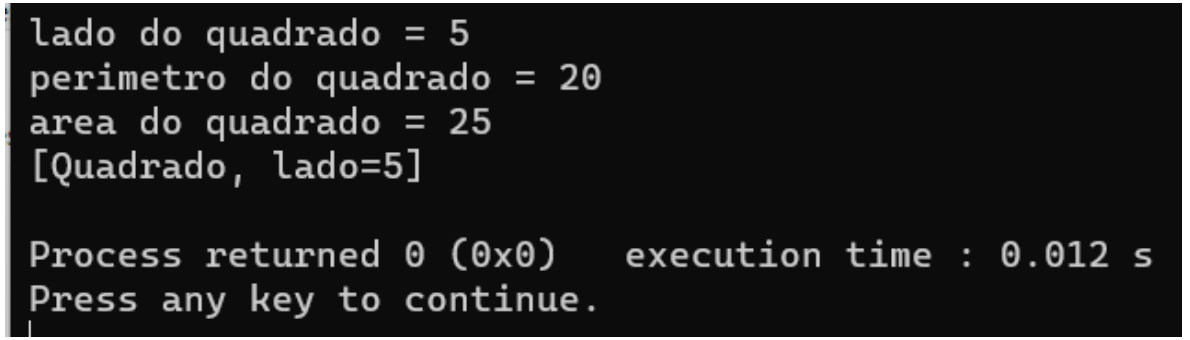
Alocação Dinâmica

- Em C++, desalocação de memória dinâmica é feita usando o operador delete e delete[].

```
delete pointer;
delete[] pointer;

int * pInt = new int; //1 inteiro
int * pInts = new int[5]; //5 inteiros

delete pInt;
delete[] pInts;
```

Alocação Dinâmica

```
int main()
{
   //Constroi quadrado de lado 5
   Quadrado *q = new Quadrado(5);

   //Imprime o lado do quadrado
   cout << "lado do quadrado = " << q->getLado() <<
endl;

   //Imprime o perimetro do quadrado
   cout << "perimetro do quadrado = " << q-
>perimetro() << endl;

   //Imprime o area do quadrado
   cout << "area do quadrado = " << q->area() << endl;

   //Imprime o quadrado
   q->print();

   //Exclui o quadrado
   delete q;
```



Alocação Dinâmica e Destrutor

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

class Quadrado {
private:
   int *lado;

public:
   Quadrado(int l) {
      lado = new int(l);
   }
   ~Quadrado() {
      delete lado;
   }
   void setLado(int lado) {
      *(this->lado) = lado;
   }
   int getLado() {
      return *lado;
   }

   int perimetro() {
      return 4*(*lado);
   }

   int area() {
      return (*lado) * (*lado);
   }

   void print() {
      cout << "[Quadrado, lado=" << *(this->lado) << "]"
<< endl;
   }
}
```

Alocação Dinâmica e Destrutor

```
int main()
{
   //Constroi quadrado de lado 100
   Quadrado q(100);

   //Imprime o lado do quadrado
   cout << "lado do quadrado = " << q.getLado() <<
endl;

   //Imprime o perimetro do quadrado
   cout << "perimetro do quadrado = " << q.perimetro()
<< endl;

   //Imprime o area do quadrado
   cout << "area do quadrado = " << q.area() << endl;

   //Imprime o quadrado
   q.print();

   return 0;
}
```

```
lado do quadrado = 100
perimetro do quadrado = 400
area do quadrado = 10000
[Quadrado, lado=100]

Process returned 0 (0x0)   execution time : 0.013 s
Press any key to continue.
```