Programação Orientada a Objetos

Prof. Delano M. Beder

Roteiro 07 – Overloading (sobrecarga) e Overiding (Sobreposição)

Representar **Triângulos** (figura geométrica de 3 lados) com as seguintes operações:

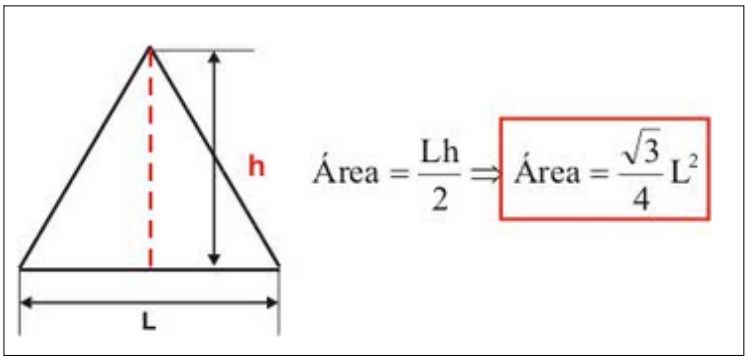
- Construtor com 3 parâmetros para inicializar os 3 lados do triângulo.
- Construtor com apenas 1 parâmetro. Nesse caso, os 3 lados do triângulo são iguais.
- Operação **getPerimetro()** que retorna o perímetro do triângulo (soma dos 3 lados)
- Operação **getArea()** que retorna a área do triângulo

> Quando conhecemos as medidas dos lados do triângulo, podemos também encontrar a área usando a seguinte fórmula:
>
> $A = \sqrt{p.(p-a).(p-b).(p-c)}$
>
> Sendo a, b e c as medidas dos lados do triângulo e p dado pela fórmula:
>
> $p = \frac{a+b+c}{2}$

- Operação **imprime** que imprime as informações de um triângulo (comprimento 3 lados, perímetro e área)
- Operação **compare** que recebe como parâmetro uma 2° triângulo **t2** e:
  - Retorna negativo se **t2** é "maior" (maior área) que o objeto que está executando a operação
  - Retorna zero se **t2** é "igual" (mesma área) ao objeto que está executando a operação
  - Retorna positivo se **t2** é "menor" (menor área) que o objeto que está executando a operação

Representar **Triângulos Equiláteros** com as seguintes operações:

- Construtor com apenas 1 parâmetro desde que os 3 lados do triângulo equilátero são iguais.
- Operação **getPerimetro()** que retorna o perímetro do triângulo (soma dos 3 lados)
- Operação **getArea()** que retorna a área do triângulo



- Operação **imprime** que imprime as informações de um triângulo (comprimento 3 lados, perímetro e área)
- Operação **compare** que recebe como parâmetro uma 2° triângulo **t2** e:
  - Retorna negativo se **t2** é "maior" (maior área) que o objeto que está executando a operação
  - Retorna zero se **t2** é "igual" (mesma área) ao objeto que está executando a operação
  - Retorna positivo se **t2** é "menor" (menor área) que o objeto que está executando a operação

1. Crie um projeto C++ denominado Triângulos

2. Implementação da abstração Triângulo

Nova classe C++ denominada Triangulo

Dois arquivos: Triangulo.h (Cabeçalho) e Triangulo.cpp (Código-fonte)

2.1 Arquivo Triangulo.h

```
#ifndef TRIANGULO_H
#define TRIANGULO_H

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

class Triangulo {
public:
  Triangulo(double lado);
  Triangulo(double lado1, double lado2, double lado3);
  double getPerimetro() const;
  virtual double getArea() const;
  void imprime() const;
  int compare(const Triangulo& t) const;
protected: // os lados do triângulo são protected (podem ser acessados pelas subclasses)
  double lados[3];
};

#endif /* TRIANGULO_H */
```

2.2 Arquivo Triangulo.cpp

```
#include "Triangulo.h"

Triangulo::Triangulo(double lado1, double lado2, double lado3) {
  lados[0] = lado1;
  lados[1] = lado2;
  lados[2] = lado3;
}

Triangulo::Triangulo(double lado) : Triangulo(lado, lado, lado) {
}

double Triangulo::getPerimetro() const {
  double soma = 0;
  for (int i = 0; i < 3; i++) {
    soma += lados[i];
  }
  return soma;
}

double Triangulo::getArea() const {
  double semi = this->getPerimetro() / 2;
  double produto = semi;
  for (int i = 0; i < 3; i++) {
    produto *= semi - lados[i];
  }
  return sqrt(produto);
}

void Triangulo::imprime() const {
  cout << "Lados: [";
  for (int i = 0; i < 3; i++) {
    cout << lados[i];
    if (i != 2) {
      cout << ",";
    }
  }
  cout << "]" << endl;
  cout << "Perímetro: " << this->getPerimetro() << endl;
  cout << "Área: " << this->getArea() << endl;
}

int Triangulo::compare(const Triangulo& t) const {
  double diff = this->getArea() - t.getArea();
  return (diff < 0) ? -1 : (diff > 0) ? 1 : 0;
}
```

3. Implementação da abstração Triângulo Equilátero

Nova classe C++ denominada Equilatero

Dois Arquivos: Equilatero.h (Cabeçalho) e Equilatero.cpp (Código-fonte)

3.1 Arquivo Equilatero.h

```
#ifndef EQUILATERO_H
#define EQUILATERO_H

#include "Triangulo.h"

class Equilatero : public Triangulo {
public:
  Equilatero(double lado);
  double getArea() const;
private:
};

#endif /* EQUILATERO_H */
```

3.2 Arquivo Equilatero.cpp

```
#include "Equilatero.h"

Equilatero::Equilatero(double lado) :
Triangulo(lado) {
}

double Equilatero::getArea() const {
  return lados[0] * lados[0] * sqrt(3) / 4;
}
```

3. Arquivo main.cpp

```
#include "Triangulo.h"
#include "Equilatero.h"

int main(int argc, char** argv) {

  Triangulo t1(3,4,5);
  Equilatero t2(3);

  t1.imprime();

  cout << endl;

  t2.imprime();

  return 0;
}
```

4. Compile e execute (verifique a saída impressa)

5. Fim