Polimorfismo

1. O mundo das formas é muito rico. Crie uma interface (em C++ uma classe abstract apenas com métodos virtuais = 0) chamada de `Forma`. A inteface mais abstrata, `Forma`, deve possuir dois métodos: `std::string get_cor()` e `std::string get_nome()`. A mesma tem duas sub-interfaces: `Forma2D` e `Forma3D`. A primeira, deve conter os métodos `double get_area()` e `double get_perimetro()`. A segunda tem um único método `double get_volume()`. Implemente as classes: `Quadrado`, `Circulo`, `Triangulo`, `Esfera` e `Cubo`. Especificamente para a classe `Triangulo` você deve criar 3 construtores, um para criação de cada tipo de triângulo, sendo eles: isósceles, equilátero e escaleno.

   Utilize o código abaixo (`main.cpp`) para testar o seu código.

```cpp
#include <iostream>

int main(){
    //lado, nome, cor
    Quadrado q(2.0, "Quadrado", "Vermelho");
    //raio, nome, cor
    Circulo c(3.0, "Circulo", "Azul");
    //lado 1, lado 2, lado 3/base, altura, nome, cor
    Triangulo t1(7.0, 3.0, 4.0, 2.0, "Triangulo Escaleno", "Amarelo");
    //lado 1/lado 2, lado 3/base, altura, nome, cor
    Triangulo t2(3.0, 4.0, 2.0, "Triangulo Isosceles", "Rosa");
    //lado 1/base, altura, nome, cor
    Triangulo t3(7.0,2.0, "Triangulo Equilatero", "Verde");
    //raio, nome, cor
    Esfera e(6.0,"Esfera", "Branco");
    //lado, nome, cor
    Cubo cb(9.0,"Cubo", "Preto");

    std::cout << "Area: " << q.get_area() << " - Perimetro: " << q.
        get_perimetro() << " - Nome: " << q.get_nome() << " - Cor: " << q.
        get_cor() << std::endl;

    std::cout << "Area: " << c.get_area() << " - Perimetro: " << c.
        get_perimetro() << " - Nome: " << c.get_nome() << " - Cor: " << c.
        get_cor() << std::endl;

    std::cout << "Area: " << t1.get_area() << " - Perimetro: " << t1.
        get_perimetro() << " - Nome: " << t1.get_nome() << " - Cor: " << t1.
        get_cor() << std::endl;

    std::cout << "Area: " << t2.get_area() << " - Perimetro: " << t2.
        get_perimetro() << " - Nome: " << t2.get_nome() << " - Cor: " << t2.
        get_cor() << std::endl;

    std::cout << "Area: " << t3.get_area() << " - Perimetro: " << t3.
        get_perimetro() << " - Nome: " << t3.get_nome() << " - Cor: " << t3.
        get_cor() << std::endl;

    std::cout << "Volume: " << e.get_volume() << " - Nome: " << e.get_nome()
        << " - Cor: " << e.get_cor() << std::endl;

    std::cout << "Volume: " << cb.get_volume() << " - Nome: " << cb.get_nome()
         << " - Cor: " << cb.get_cor() << std::endl;
}
```

2. Uma farmácia necessita controlar todos os produtos que comercializa. Sabe-se que nesse estabelecimento os produtos comercializados são medicamentos e artigos de higiene. Os medicamentos possuem código, descrição, preço de compra, percentual de lucro e valor do desconto do laboratório. Os artigos de higiene possuem código, descrição, preço de compra e percentual de lucro. Sabe-se que um artigo de higiene gera como lucro 30% do preço de compra.

   - Crie as classes que representem o contexto descrito;
   - Crie na classe Remedio o seguinte método:
     - `gerarPrecoDeVenda`: método sem parâmetros que deverá retornar o preço de venda do remédio. O preço de venda é a soma do preço de compra mais o percentual de lucro do preço de compra, menos o desconto do laboratório.
   - Crie na classe ArtigoHigiene o seguinte método:
     - `gerarPrecoDeVenda`: método sem parâmetros que deverá retornar o preço de venda do artigo de higiene. O preço de venda é a soma do preço de compra mais o percentual de lucro do preço de compra.
   - Crie um `main.cpp`. Crie dois remédios e dois artigos de higiene. Utilize todos os métodos das classes criadas. Imprima todos os dados dos produtos. Importante! Para resolução deste exercício você deve empregar, obrigatoriamente, os conceitos de herança e polimorfismo.

     Utilize o código abaixo (`main.cpp`) para testar o seu código.

```cpp
#include <iostream>

int main(){
    Remedio r1(123, "Amoxilina", 500.0, 5.0, 10);
    Remedio r2(456, "Ibuprofeno", 2000.0, 7.0, 20);

    ArtigoHigiene ah1(789, "Desodorante", 1500.0);
    ArtigoHigiene ah2(112, "Shampoo", 800.0);

    std::cout << "Codigo: " << r1.get_codigo() << " - Descricao: " << r1.
        get_descricao() << " - % de Lucro: " << r1.get_percentualDeLucro()
        << " - Preco de Compra: " << r1.get_precoDeCompra() << " - Valor do
         desconto: " << r1.get_valorDoDesconto() << " - Preco de venda: "
        << r1.gerarPrecoDeVenda() << std::endl;
    std::cout << "Codigo: " << r2.get_codigo() << " - Descricao: " << r2.
        get_descricao() << " - % de Lucro: " << r2.get_percentualDeLucro()
        << " - Preco de Compra: " << r2.get_precoDeCompra() << " - Valor do
         desconto: " << r2.get_valorDoDesconto() << " - Preco de venda: "
        << r2.gerarPrecoDeVenda() << std::endl;

    std::cout << "Codigo: " << ah1.get_codigo() << " - Descricao: " << ah1
        .get_descricao() << " - % de Lucro: " << ah1.get_percentualDeLucro
        () << " - Preco de Compra: " << ah1.get_precoDeCompra() << " -
        Preco de venda: " << ah1.gerarPrecoDeVenda() << std::endl;
    std::cout << "Codigo: " << ah2.get_codigo() << " - Descricao: " << ah2
        .get_descricao() << " - % de Lucro: " << ah2.get_percentualDeLucro
        () << " - Preco de Compra: " << ah2.get_precoDeCompra() << " -
        Preco de venda: " << ah2.gerarPrecoDeVenda() << std::endl;
}
```

Considerações Gerais!

- Exercício individual.
- Entrega: conforme agendado no PVANET Moodle;

- Conforme estrutura abaixo apresentada crie um projeto para resolução de cada exercício (ex.: pratica6_exercicio1.zip, pratica6_exercicio2.zip, etc). Cada projeto deve conter os arquivos `.h`, `.cpp`, e `main.cpp` criados para resolução do exercício. Envie, através do PVANet Moodle, uma pasta compactada (.rar ou .zip) contendo todos os projetos (também compactados). A pasta compactada deve conter informações do aluno (ex.: julio_reis-pratica6.zip).

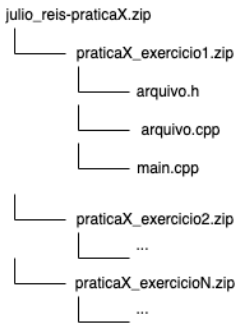

Figura 1: Estrutura de diretórios.

- O seu `main.cpp` deve conter, minimamente, instruções para criação (instanciação de objetos) e chamadas das funções implementadas (TODAS!!!). Para teste, você pode usar os exemplos fornecidos.