Herança e Composição

1. Crie uma classe `SavingsAccount`. Utilize um membro de dados `static annualInterestRate` para armazenar a taxa de juros anual para cada um dos correntistas. Cada membro da classe contém um membro de dados `private savingsBalance` para indicar a quantia que os correntistas têm atualmente em depósito. Forneça a função-membro `calculateMonthlyInterest` que calcula os juros mensais multiplicando o `savingsBalance` pelo `annualInterestRate` dividido por 12; esses juros devem ser adicionados a `savingsBalance`. Forneça uma função-membro `static modifyInterestRate` que configura o `static annualInterestRate` com um novo valor. Escreva um programa `main` para testar a classe `SavingsAccount`. Instancie dois objetos diferentes da classe `SavingsAccount`, `saver1` e `saver2`, com saldos de $ 2.000,00 e $ 3.000,00, respectivamente. Configure o `annualInterestRate` como 3%. Em seguida, calcule os juros mensais e imprima os novos saldos de cada um dos correntistas. Então configure novamente o `annualInterestRate` como 4%, calcule os juros do próximo mês e imprima os novos saldos para cada um dos poupadores.

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

```cpp
int main(void) {
    // Main. Criando duas contas.
    SavingsAccount saver1 = SavingsAccount(2000);
    SavingsAccount saver2 = SavingsAccount(3000);

    // Imprimindo o monthly balance
    std::cout << saver1.calculateMonthlyInterest() << std::endl;
    std::cout << saver2.calculateMonthlyInterest() << std::endl;

    // Alterando atributo static publico
    SavingsAccount::annualInterestRate = 3.00;

    // Imprimindo o monthly balance. Tem que mudar para as duas classes.
    std::cout << saver1.calculateMonthlyInterest() << std::endl;
    std::cout << saver2.calculateMonthlyInterest() << std::endl;

    return 0;
}
```

2. Analise o código disponibilizado nos slides da aula (hierarquia de classes: `Pessoa` e `Estudante`), entenda-o, e depois execute-o para ver/analisar os resultados. Em seguida:

   - Crie a Classe `Turma`, com atributos privados código (String), e ano (int). Crie um construtor que receba parâmetros para inicializar os atributos e os métodos de acesso e métodos modificadores (gets e sets).
   - Altere a classe `Estudante` para que tenha também um atributo privado turma do tipo `Turma`. Altere o construtor para receber um parâmetro que inicialize o novo atributo e crie os métodos de acesso e modificador para este novo atributo.
   - Altere o main para tratar esse novo atributo da classe `Estudante`.

```cpp
#include <iostream>

#include "estudante.h"
#include "pessoa.h"

```

```
void f(Pessoa &pessoa) {
  std::cout << "Na funcao: " << pessoa.defina_meu_tipo() << std::endl;
}

int main() {
  Pessoa pessoa("Julio Reis.");
  Estudante estudante("Jane Doe", 20180101);
  std::cout << "A pessoa eh: " << pessoa.defina_meu_tipo() << std::endl;
  std::cout << "O estudante eh: " << estudante.defina_meu_tipo() << std::endl;
  f(pessoa);
  f(estudante);

  return 0;
}
```

3. Os serviços de correio expresso, como FedEx, DHLe UPS, oferecem várias opções de entrega, cada qual com custos específicos. Crie uma hierarquia de herança para representar vários tipos de pacotes. Utilize `Package` como a classe básica da hierarquia, então inclua as classes `TwoDayPackage` e `OvernightPackage` que derivam de `Package`. A classe básica `Package` deve incluir membros de dados que representam nome, endereço. Para simplificar nosso código, represente o endereço como uma única string. Além dos membros citados anteriormente, armazene dados que representam o peso (em quilos) e o custo por quilo para a entrega do pacote. O construtor `Package` deve inicializar esses membros de dados, em outras palavras, todos são argumentos do construtor. Assegure que o peso e o custo por quilo contenham valores positivos (faça uso de unsigned `int`). `Package` deve fornecer um método `public calculateCost` que retorna um `double` indicando o custo associado com a entrega do pacote. A função `calculateCost` de `Package` deve determinar o custo multiplicando o peso pelo custo (em quilos). A classe derivada `TwoDayPackage` deve herdar a funcionalidade da classe básica `Package`, mas também incluir um membro de dados que representa uma taxa fixa que a empresa de entrega cobra pelo serviço de entrega de dois dias. O construtor `TwoDayPackage` deve receber um valor para inicializar esse membro de dados. `TwoDayPackage` deve redefinir a função-membro `calculateCost` para que ela calcule o custo de entrega adicionando a taxa fixa ao custo baseado em peso calculado pela função `calculateCost` da super classe `Package`. A classe `OvernightPackage` deve herdar diretamente da classe `Package` e conter um membro de dados adicional para representar uma taxa adicional por quilo cobrado pelo serviço de entrega noturno. `OvernightPackage` deve redefinir a função-membro `calculateCost` para que ela acrescente a taxa adicional por quilo ao custo-padrão por quilo antes de calcular o custo da entrega.

```
#include <iostream>
#include "Package.h"
#include "TwoDayPackage.h"
#include "OvernightPackage.h"

int main(){
    Package package("Pacote 1", "Rua Passos, 71", 20, 15);
    TwoDayPackage two_day_package("Pacote 2", "Av. PH Holfs, s/n", 5, 15, 10);
    OvernightPackage over_night_package("Pacote 3", "Av. Santa Rita, 110", 50,
        15, 15);

    std::cout << package.calculateCost() << std::endl;
    std::cout << two_day_package.calculateCost() << std::endl;
    std::cout << over_night_package.calculateCost() << std::endl;

    return 0;
}
```

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).

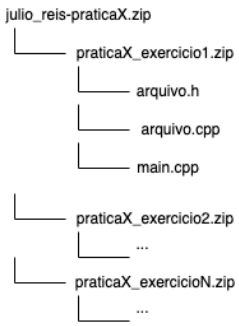

Figura 1: Estrutura de diretórios.

- 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.