|  |  |  |
| --- | --- | --- |
| Logo_UFOP | **Universidade Federal de Ouro Preto – UFOP**  **Instituto de Ciências Exatas e Biológicas – ICEB**  **Departamento de Computação – DECOM**  **Disciplina: Programação Orientada a Objetos**  **Professor: Marco Antonio M. Carvalho** | decom_novo |

**Lista de Exercícios 03 – Classes, Objetos, Métodos, Sobrecarga, Funções Amigas**

**Instruções**

* *Todos os exercícios que envolvem programas devem ser resolvidos por programas em linguagem* ***C++****;*
* *Na solução dos exercícios, devem ser utilizados os conceitos listados no cabeçalho desta lista;*
* *Eventuais dúvidas podem ser sanadas com o professor.*

1. Modifique a classe *GradeBook* como a seguir. Modifique também o programa para testar a nova classe.
   1. Inclua um segundo atributo *string* que represente o nome do professor da disciplina;
   2. Crie um *getter* e um *setter* para este novo atributo;
   3. Modifique o construtor para receber dois parâmetros, um para o nome da disciplina e outro para o nome do professor;
   4. Modifique o método *displayMessage* para apresentar o nome do professor também.
2. Crie uma classe chamada *Conta*, que represente contas bancárias. A classe deve conter como atributo o saldo da conta (um número real). A classe deve possuir um construtor que recebe o saldo inicial para inicializar o atributo, validando se o valor é maior ou igual a zero. Caso o valor seja menor a zero, o atributo deve ser inicializado com zero e uma mensagem de erro deve ser apresentada. Crie um programa que contenha dois objetos desta classe e utilize cada um dos três métodos:
   1. *Credito*: adiciona um valor ao saldo atual;
   2. *Debito*: subtrai um valor do saldo atual, garantindo que o saldo não ficará negativo. Se o débito for maior que o saldo, a operação não deve ser realizada e uma mensagem apresentada;
   3. *getSaldo*: retorna o saldo.
3. Crie uma classe chamada *NotaFiscal* que um *hardware* utilize para representar uma nota fiscal em uma loja de peças. Uma nota fiscal deve incluir quatro dados como atributos:
   1. Número da peça (*string*);
   2. Descrição da peça (*string*);
   3. Quantidade comprada (inteiro);
   4. Preço (número real).

A classe deve incluir ainda *getters* e *setters* para cada um dos atributos. Ainda, deve haver um método *getTotalNota* que calcule e retorne o total de um vetor de objetos. Escreva um programa que teste cada um dos métodos da classe.

1. Crie uma classe *Empregado* que inclua os atributos nome (*string*), sobrenome (*string*) e salário mensal (número real). A classe deve incluir construtores para cada um dos atributos para evitar lixo e também *getters* e *setters*. Crie um programa que teste a classe, criando dois objetos, calculando o salário anual de cada, dando um aumento de 10% e calcule o salário anual novamente.
2. Crie uma classe *Data* que inclua três atributos: mês (inteiro), dia (inteiro) e ano (inteiro). Crie métodos para:
   1. Funcionar como *getter* e *setter*, validando os dados para garantir que os valores são reais;
   2. Funcionar como um construtor (com parâmetros padronizados) com três parâmetros para inicialização dos três atributos;
   3. Imprimir a data, com os campos separados por /;
   4. Calcular a quantidade de dias do ano até aquele mês, recebendo como parâmetro o número do mês;
   5. Sobrecarregue o método anterior para realizar o mesmo cálculo, porém, recebendo como parâmetro o nome do mês, em letras minúsculas.

Escreva um programa que teste todos os métodos da classe. Para simplificar o exercício, considere meses ímpares com 31 dias e meses pares com 30 dias.

1. Crie uma classe *Aluno* com atributos que armazenem o nome, a série que cursa e o grau. Crie um *getter* e um *setter*, além de uma variável *static* que conte a quantidade de objetos criados, a ser utilizada no construtor e destrutor. Crie um vetor com 15 objetos desta classe e preencha os dados através do *setter*.
2. Crie uma classe *Tempo* com três atributos: horas, minutos e segundos. Crie dois construtores: um para inicializar os atributos com valor 0 e outro para inicializar os atributos com valores passados como argumentos. Crie métodos para:
   1. Funcionar como *getter* e *setter;*
   2. Imprimir os atributos no formato hh:mm:ss;
   3. Subtrair dois objetos e colocar o resultado no objeto que o chamou;
   4. Somar que soma dois objetos e colocar o resultado no objeto que o chamou;
   5. Sobrecarregue este último método para que retorne um objeto com o resultado da operação.
3. Crie uma classe *Estacionamento* para armazenar dados de um estacionamento. Os atributos devem representar a placa e modelo do carro além da hora de entrada e saída do estacionamento. Utilize dois objetos da classe *Tempo* criada no exercício anterior. Crie métodos para:
   1. Funcionar como *getter* e *setter;*
   2. Inicializar os dados com vazio ou zero;
   3. Imprimir os dados de um carro estacionado;
   4. Calcular e retornar o valor a ser pago pelo carro estacionado. Considere o preço de R$1,50 por hora. Utilize o método da classe *Tempo*.

Crie um vetor de 5 posições e preencha-o para testar sua classe.

1. Crie uma classe *NumeroRacional* (fração), com as seguintes características
   1. Crie um construtor que evita o valor zero ou negativo como denominador;
   2. Sobrecarregue os operadores de adição, subtração, multiplicação e divisão entre objetos desta classe;
   3. Sobrecarregue os operadores relacionais para os objetos desta classe.
2. Sobrecarregue os operadores de adição, subtração, multiplicação e divisão entre objetos da classe criada no exercício anterior e **números inteiros**.
3. Considere a classe *NumeroComplexo* definida abaixo. Os números complexos são representados na forma *parteReal*+*parteImaginaria* \**i*, em que *i* = . Sobrecarregue o operador de multiplicação **\*** de dois números complexos e os operadores relacionais **==** e **!=**. Utilize o driver também definido abaixo para testar sua classe.

***complex.h***

**class** Complex

**{**

**public:**

Complex**(** **double** **=** 0.0**,** **double** **=** 0.0 **);** // construtor

Complex **operator+(** **const** Complex **&** **)** **const;** // adição

Complex **operator-(** **const** Complex **&** **)** **const;** // subtração

**void** print**()** **const;** // saída

**private:**

**double** real**;** // parte real

**double** imaginary**;** // parte imaginária

**};** // fim da classe Complex

***complex.cpp***

#include <iostream>

**using** namespace std**;**

#include "Complex.h" // definição da classe Complex

// Construtor

Complex**::**Complex**(** **double** realPart**,** **double** imaginaryPart **)**

**:** real**(** realPart **),** imaginary**(** imaginaryPart **)**

**{**

// corpo vazio

**}** // fim do construtor Complex

// operador de adição

Complex Complex**::operator+(** **const** Complex **&**operand2 **)** **const**

**{**

**return** Complex**(** real **+** operand2**.**real**,**

imaginary **+** operand2**.**imaginary **);**

**}** // fim da função operator+

// operador de subtração

Complex Complex**::operator-(** **const** Complex **&**operand2 **)** **const**

**{**

**return** Complex**(** real **-** operand2**.**real**,**  imaginary **-** operand2**.**imaginary **);**

**}** // fim da função operator-

// exibe um objeto Complex na fórmula: (a, b)

**void** Complex**::**print**()** **const**

**{**

cout **<<** '(' **<<** real **<<** ", " **<<** imaginary **<<** ')'**;**

**}** // fim da função print

***driverComplex.cpp***

// Programa de teste da classe Complex.

#include <iostream>

**Using namespace** std**;**

#include "Complex.h"

**int** main**()**

**{**

Complex x**;**

Complex y**(** 4.3**,** 8.2 **);**

Complex z**(** 3.3**,** 1.1 **);**

cout **<<** "x: "**;**

x**.**print**();**

cout **<<** "\ny: "**;**

y**.**print**();**

cout **<<** "\nz: "**;**

z**.**print**();**

x **=** y **+** z**;**

cout **<<** "\n\nx = y + z:" **<<** endl**;**

x**.**print**();**

cout **<<** " = "**;**

y**.**print**();**

cout **<<** " + "**;**

z**.**print**();**

x **=** y **-** z**;**

cout **<<** "\n\nx = y - z:" **<<** endl**;**

x**.**print**();**

cout **<<** " = "**;**

y**.**print**();**

cout **<<** " - "**;**

z**.**print**();**

cout **<<** endl**;**

**return** 0**;**

**}** // fim de main