Pontifícia Universidade Católica de Goiás
Escola de Ciências Exatas e da Computação
CMP1048 - Técnicas de Programação
Max Gontijo de Oliveira
Lista de Exercícios 3:A - Classes

Boas práticas para codificação de classes.

1. Arquivo .h

```
...
/* Em um arquivo .h (por exemplo, minha_lib.h), DECLARE todas as
   classes (se houver) sem implementar nenhuma */
class Classe_A;
class Classe_B;

/* No mesmo arquivo .h, DECLARE todas as funções (se houver) sem
   implementar nenhuma */
void func_1();
float func_1(int x);
int func_2(Classe_A obj_a);
int func_2(Classe_A obj_a, Classe_A obj_b);

/* No mesmo arquivo .h, DEFINA todas as classes declaradas (não
   implemente os métodos nem construtores) */
class Classe_A
{
   public:
      int a;
      float b;
      Classe_A(int v); // construtor
      void metodo_1();
      char metodo_2(int x);
      void metodo_3(char* x, int n);
};

class Classe_B
{
   public:
      float c;
      char* b;
      int metodo_1();
      float metodo_4(int x, int y);
};
...
```

2. Arquivo .cpp

```
...
/* Em um arquivo .cpp (por exemplo, minha_lib.cpp), DEFINA (implemente)
   todas as funções e todos os métodos de todas as classes,
   independente da ordem */
void func_1()
{
    ...
}

float func_1(int x)
{
    ...
}

int func_2(Classe_A obj_a)
{
    ...
}

...

/* A definição de métodos segue o seguinte formato:
   <RETORNO> <NOME_CLASSE>::<NOME_MÉTODO>(<PARÂMETROS>) { ... } */
void Classe_A::metodo_1()
{
    ...
}

char Classe_A::metodo_2(int x)
{
    ...
}

...

float Classe_B::metodo_4(int x, int y)
{
    ...
}

/* A definição dos construtores seguem um formato parecido, mas não tem
   retorno e, no lugar no nome do método, simplesmente repete-se o nome
   da classe:
   <NOME_CLASSE>::<NOME_CLASSE>(<PARÂMETROS>) { ... } */
Classe_A::Classe_A(int v)
{
    ...
}
...
```

Consideração sobre strings e datas:

Nessa lista de exercícios, quando for mencionado que o tipo de uma variável é string, o aluno deverá considerar o tipo `string` do C++, o ponteiro para vetor de `char` (`char*`) ou um vetor de `char`

propriamente dito. A escolha de qual tipo utilizar será do próprio aluno.

Quando for mencionado que o tipo de uma variável é uma data, o aluno deverá considerar que se trata de uma string no formato ***DD/MM/AAAA***.

1. Crie uma classe denominada `Elevador` para armazenar as informações de um elevador dentro de um prédio. A classe deve armazenar o andar atual (0 = térreo e não existe subsolo), total de andares no prédio, excluindo o térreo, capacidade do elevador, e quantas pessoas estão presentes nele. Defina um construtor que receba a quantidade de andares do prédio e a capacidade do elevador. A classe deve também disponibilizar os seguintes métodos:

   - `entra` → recebe o número de pessoas que entra no elevador.
   - `sai` → recebe o número de pessoas que sai do elevador.
   - `sobe` → recebe o número andares que irá subir a partir do local atual.
   - `desce` → recebe o número andares que irá descer a partir do local atual.

   Não se esqueça de fazer tratamentos para o caso de querer entrar mais pessoas do que a capacidade, sair mais pessoas do que tem dentro do elevador, subir ou descer mais andares do que é possível.

   Crie um programa que teste a sua classe.

2. Crie uma classe em C++ chamada `Relogio` para armazenar um horário, composto por hora, minuto e segundo. A classe deve representar esses componentes de horário e deve apresentar os métodos descritos a seguir:

   - `set_horario` → define o horário do relógio por meio de três parâmetros inteiros (hora, minuto, segundo).
   - `get_horario` → retorna hora, minuto e segundo por meio de três parâmetros passados por referência.
   - `avancar_1_segundo` → avança um segundo no horário atual do relógio.
   - `imprimir` → imprime o horário do relógio com o formato `"HH:MM:SS"`.

   Crie um programa que instancia um `Relogio` e inicializa seu horário. Em seguida, faça um laço repetitivo com 200 iterações que, em cada iteração, chama o método `avancar_1_segundo` e imprime o horário atual.

3. Crie uma classe chamada `CanalTV` que tenha dois atributos: o número do canal e o nome do canal. Crie uma outra classe chamada `TV` que tenha apenas quatro atributos: uma variável de situação do tipo `bool` que indica se a TV está ligada (onde o valor `true` indica que está ligada e o valor `false` indica que está desligada), quantidade de canais existentes, canal selecionado (não é a posição do canal no vetor, mas o número efetivo do canal) e um vetor de canais (pode ser fixo `CanalTV canais[5]`; ou dinâmico `CanalTV* canais`). Na classe `TV`, crie ainda os seguintes métodos:

   - `ligar_desligar` → não recebe parâmetros e altera a situação da TV de ligada para desligada ou de desligada para ligada.
   - `tv_ligada` → não recebe parâmetros e retorna `true` caso a TV esteja ligada ou `false` caso contrário.
   - `numero_canal_atual` → se a TV estiver ligada, retorna o número do canal atualmente selecionado. Se estiver desligada, retorna -1.
   - `nome_canal_atual` → se a TV estiver ligada, retorna o nome do canal atualmente selecionado. Caso o canal selecionado seja um número de um canal que não existe no vetor de canais, deverá retornar o texto `"Sem sinal"`. Se estiver desligada retorna `"TV está desligada"`.
   - `avancar_canal` → se a TV estiver ligada, altera o canal atual para o número do canal EXISTENTE no vetor de canais imediatamente maior do que o canal atual. Caso o canal atual seja o maior número entre os canais existentes no vetor de canais, o próximo deverá ser aquele com

o menor número. Caso o número do canal atual seja um número que nenhum canal no vetor de canais possui, a regra será a mesma: avança para o imediatamente maior após o número do canal atual. O método não precisa retornar nada. Se a TV não estiver ligada, não faz nada.

- `retroceder_canal` → faz a mesma coisa que o método `avancar_canal`, mas ao invés de avançar para o canal imediatamente maior, retrocede ao canal imediatamente menor.
- `ir_para_canal` → recebe um número inteiro que é o canal que deseja-se acessar diretamente. Se a TV estiver ligada, o canal atual passa a ser esse, independente de ele existir no vetor de canais. Mas o método deve retornar `true` caso o canal exista ou `false` caso não exista no vetor de canais. Caso a TV esteja desligada, não deve fazer nada e retornar `false`.
- `imprimir` → Se a TV estiver ligada, imprime o número e o nome do canal selecionado. Caso o canal selecionado não exista no vetor de canais, deve-se imprimir `"Sem sinal"` no lugar do nome do canal. Se a TV estiver desligada, imprime `"TV está desligada"`.

4. Crie uma classe chamada `Retangulo` que tenha apenas dois atributos inteiros: base e altura. Crie um construtor para a classe `Retangulo` que receba essa base e essa altura. Agora, na classe `Retangulo` crie os seguintes métodos:

   - `area` → retorna a área do retângulo.
   - `imprimir` → imprime o retângulo no terminal utilizando apenas o caractere `#` de forma proporcional ao tamanho de até 50 colunas.
     - Exemplo 1: base é 20 e a altura é 30.
       Como a base é quem define a quantidade de colunas no terminal e como 20 é menor do que 50, deverá ser impresso no terminal 30 linhas contendo 20 caracteres `#` cada linha.
     - Exemplo 2: base é 70 e a altura é 20.
       Como a base é quem define a quantidade de colunas no terminal e como 70 é maior do que 50, deverá ser impresso no terminal 14 linhas contendo 50 caracteres `#` cada linha. Esse número de linhas é obviamente deduzido a partir do arredondamento do resultado da multiplicação de 20 por $50 \div 70$, que é a proporção que deve ser considerada ao reduzir o tamanho da base de 70 para 50.
   - `imprimir` → uma sobrecarga do método anterior que recebe o caractere a ser impresso.
   - `imprimir` → uma sobrecarga dos métodos anteriores, mas que recebe o caractere a ser impresso no contorno e o caractere a ser impresso no interior do retângulo.

   Crie um programa para testar sua classe.

5. Crie uma classe chamada `Ponto` que tenha apenas dois atributos float: as coordenadas x e y. Agora, crie uma classe chamada `Triangulo` que tenha apenas três atributos do tipo `Ponto`: A, B e C. Crie um construtor para a classe `Triangulo` que receba os três pontos que o definem. Agora, na classe `Triangulo` crie os seguintes métodos:

   - `lado_AB` → retorna o tamanho do lado formado pelos pontos A e B do triângulo.
   - `lado_AC` → retorna o tamanho do lado formado pelos pontos A e C do triângulo.
   - `lado_BC` → retorna o tamanho do lado formado pelos pontos B e C do triângulo.
   - `tipo_triangulo` → retorna 1, se o triângulo é equilátero, 2 se o triângulo é isósceles, 3 se o triângulo é escaleno ou -1 se os pontos não formam um triângulo (caso em que pelo menos dois pontos são iguais).

   Crie um programa para testar sua classe.

6. Crie uma classe chamada `Arma` que tenha dois atributos inteiros que representem a capacidade e a quantidade de munição disponível. Os atributos devem ser privados. Crie nessa classe:

   - Métodos *get* para os atributos.
   - Construtor que não receba parâmetros e inicialize a capacidade com o valor 8.

- Construtor que receba apenas um parâmetro inteiro e inicialize a capacidade com esse valor.
- Construtor que receba dois parâmetros inteiros e inicialize a capacidade e a munição com esses valores.
- Um método chamado `disparar` sem parâmetros que, se houver munição na arma, decrementa a quantidade de munição e retorna `true`; ou simplesmente retorna `false` caso não haja munição.
- Um método chamado `recarregar` sem parâmetros que faz com que a munição receba o máximo da capacidade.
- Um método chamado `recarregar` que recebe a quantidade de munição que deverá ser utilizada para recarregar a arma. O método deverá retornar um inteiro que representa a quantidade de munição que por ventura tenha sobrado após o recarregamento. Evidentemente, se a quantidade passada por parâmetro for menor ou igual do que a quantidade de munição que falta para recarregar totalmente a arma, o método deverá retornar 0.

7. Crie um programa que leia do teclado dois números inteiros e instancie duas armas (classe `Arma` da questão 6) com as capacidades dadas pelos dois números inteiros lidos. Seu programa deverá entrar em um *loop* infinito pedindo ao usuário que escolha uma dentre 4 opções: disparar com a arma 1; disparar com a arma 2; recarregar a arma 1; recarregar a arma 2. Toda vez que o usuário escolher uma ação, o programa deverá executar o solicitado e imprimir o status das armas (capacidade e munição restante). Para as opções de disparo, o programa deverá imprimir, ainda, a palavra "BANG"se disparo tiver sido dado com sucesso (tinha munição), ou "CLICK"caso o disparo não tenha funcionado (arma estava descarregada). Para as opções de recarregamento, o programa deve informar qual arma foi recarregada. Após executar a ação selecionada, o programa deve votar a pedir para o usuário escolher uma ação novamente. O programa deve finalizar quando o usuário escolher uma quinta ação, chamada SAIR.