



INSTITUTO FEDERAL DA PARAÍBA
CAMPUS CAMPINA GRANDE
BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO
DISCIPLINA DE POO e LAB. POO
PROF. VICTOR ANDRÉ PINHO DE OLIVEIRA

Atividade Unidade II - 2 - Classes (Parte 1)

Instruções

Responda às questões abaixo. Pode usar este próprio documento. Questões práticas devem ser anexadas separadamente.



Legenda:

Quirrell: Questões mais simples e diretas

Olho-Tonto: Questões de nível intermediário

Dolores: Questões de nível mais difícil, exigentes

Voldemort: Questões com nível de exigência altíssimo, desafiadoras

Questões

1. **Quirrell** O que é coesão no contexto da Orientação a Objetos?

R: É o grau em que os métodos de uma classe estão relacionados entre si e ao propósito da classe.

2. **Quirrell** Qual a utilidade dos métodos inline?

R: Reduzir o tempo de execução evitando chamadas de função, inserindo o código diretamente onde o método é usado.

3. **Quirrell** Por que devemos qualificar um método como const?

R: Para garantir que o método não altere os dados do objeto.

4. **Quirrell** O que acontece se um objeto const tentar invocar um método não-const? Explique.

R: Gera erro de compilação, pois métodos não-const podem modificar o objeto.

5. **Quirrell** O que é e para que serve a sintaxe de inicialização de membros?

R: É usada para inicializar atributos diretamente na criação do objeto, antes da execução do corpo do construtor.

6. **Quirrell** O que acontece quando atribuímos um objeto a outro (sendo eles do mesmo tipo)?

R: É feita uma cópia membro a membro (cópia rasa).

7. **Quirrell** O que acontece quando definimos métodos private? Qual a utilidade?

R: Eles só podem ser acessados dentro da própria classe; útil para ocultar detalhes internos.

8. **Quirrell** O que é, para que serve e quando é invocado o destrutor de uma Classe?

R: É um método especial que libera recursos; é invocado automaticamente quando o objeto é destruído.

9. **Quirrell** Quais as implicações de um método get retornar uma referência não-const para um atributo private?

R: Permite que o valor do atributo seja alterado fora da classe, quebrando o encapsulamento.

10. **Quirrell** (Classe *Pessoa*) Crie uma Classe *Pessoa* com atributos *nome*, *CPF* e *CPFvalido*. Defina um único construtor que receba os três argumentos. Utilize argumentos *default* para *CPF* e *CPFvalido*. Forneça também métodos *get* e *set* para

os atributos. Considere que o CPF seja passado como uma sequência de 11 dígitos, sem pontos e sem traços, no tipo string. Escreva também um método chamado apresentar, que exibe na tela o nome e o CPF. Ao exibir o CPF, deverá constar entre parênteses a situação do CPF (“Válido” ou “Inválido”). Escreva um programa de teste que demonstre as capacidades da sua classe.

11. **Olho-Tonto** (Classe *Retangulo*) Crie uma Classe Retangulo com os atributos altura e largura, cada um dos quais assume o padrão 1.0 (no construtor). Forneça métodos get e set para esses atributos. Os métodos set devem verificar se o valor passado é maior que 0.0 e menor que 20.0. Caso não seja, deve ser atribuído o valor 1.0 ao atributo. Além disso, forneça métodos que:
- Calcula e retorna o perímetro do retângulo
 - Calcula e retorna a área do retângulo
 - Desenhe o retângulo na proporção de sua largura e altura. Ex. de uma saída de largura 5 e altura 3 (as bordas não contam):

```
+-----+
|       |
|       |
|       |
+-----+
```

12. **Olho-Tonto** (Aprimorando a Classe *Time*) Modifique a Classe Time da última versão em “sala” ([ex06-atribuicao](#)) para incluir um método tick que incrementa 1 segundo à hora. O objeto Time sempre deve permanecer em um estado consistente. Escreva um programa que testa o seu método tick. Certifique-se de testar os seguintes casos:
- Incrementar para o próximo minuto. Ex.: 11h50m59 + 1s -> 11h51m00
 - Incrementar para a próxima hora. Ex.: 11h59m59 + 1s -> 12h00m00
 - Incrementar para o próximo dia. Ex.: 23h59m59 + 1s -> 0h00m00
13. **Dolores** (Aprimorando a Classe *Time*) Forneça um construtor que seja capaz de utilizar a hora atual do sistema para inicializar um objeto da Classe Time. Para tanto, siga os seguintes passos:
- Use a última versão de “sala” da Classe Time ([ex06-atribuicao](#))
 - Remova o argumento default mais a esquerda (da hora) do construtor
 - Adicione um novo construtor sem parâmetros (sobrecarregando o construtor) que inicializa o objeto com a hora atual

- 4) Pesquise como pegar a hora atual do sistema a partir da função `time()` da biblioteca `<ctime>`. Se conseguir descobrir com uma biblioteca mais atual de C++, eu agradeço 😊.

14. **Dolores** (Classe *Pessoa Revisitada*) Modifique a Classe *Pessoa* criada anteriormente (Questão 10 desta lista) de modo a atender aos seguintes requisitos:

- deve possuir os mesmos três atributos: nome, CPF e CPFvalido;
- Você deverá definir apenas um construtor e este deverá receber apenas o nome e o CPF;
- Forneça métodos `get` e `set` apenas para os dois primeiros atributos;
- O método `setCPF` deverá validar o CPF (acesse este [link](#) para conhecer o algoritmo de validação) e, em função disso, setar o valor para o atributo `CPFvalido` (`true` ou `false`);
- Mantenha o método chamado `apresentar` como já definido anteriormente.

15. **Dolores** (Classe *ViraTempo - Hogwarts*) Escreva uma Classe chamada *ViraTempo*. Para os propósitos da questão, pense nela como uma ampulheta, capaz de contar de 0 a N, indo ou voltando na contagem a partir de qualquer número. Modele-a de modo que a classe tenha os seguintes métodos principais:

- `mostrarTempo()`: vai exibir o tempo (número atual);
- `avancarTempo()`: vai incrementar ou decrementar a contagem. Não deve permitir que os limites (0 e N) do *ViraTempo* sejam ultrapassados;
- `virarTempo()`: vai controlar se o próximo avanço de tempo vai incrementar ou decrementar;

Pense nos atributos necessários para atender os requisitos da questão. Implemente um construtor que melhor atenda à sua implementação. Acrescente outros métodos se achar necessário.

Use a função **main** abaixo para validar sua resposta:

```
int main() {
    ViraTempo vt(5);
    vt.mostrarTempo();
    vt.avancarTempo();
    vt.mostrarTempo();
    vt.virarTempo();
    vt.avancarTempo();
    vt.mostrarTempo();
    vt.virarTempo();
    for (int i = 0; i < 6; i++){
        vt.avancarTempo();
    }
}
```

```

        vt.mostrarTempo();
    }
    vt.virarTempo();
    for (int i = 0; i < 7; i++){
        vt.avancarTempo();
        vt.mostrarTempo();
    }

    return 0;
}

```

A saída esperada é:

```

0
1
0
1
2
3
4
5
5
4
3
2
1
0
0
0

```

16. **Voldemort** (Classe *HugeInteger*) Crie uma Classe *HugeInteger* que utiliza um array C++ de 40 elementos (char) para armazenar “inteiros” de até 40 dígitos. A ideia é que cada elemento guarde um algarismo do número inteiro. Portanto, $1 \leq \text{len}(\text{num}) \leq 40$.

Forneça os métodos:

- input**: recebe uma string contendo o “inteiro”. O método deve verificar se realmente está recebendo um número. Não precisa considerar sinal (+ ou -), pois os números quando corretos sempre serão positivos.
- output**: imprime o número na saída padrão
- add**: soma objeto com outro *HugeInteger*
- isEqualTo**, **isNotEqualTo**, **isGreaterThan**, **isLessThan**, **isGreaterThanOrlqual** e **isLessThanOrlqual** (`==`, `!=`, `>`, `<`, `>=`, `<=`): compara o

objeto com outro objeto HugelInteger passado como argumento. O retorno deve ser do tipo bool.

Dicas:

- Você pode armazenar o número de forma invertida, corrigindo na hora de exibir. Apenas uma opção, você pode encontrar um jeito melhor de fazer.
- Você pode definir um atributo para armazenar o tamanho em dígitos do número. Isso vai te ajudar na hora de comparar os objetos.
- Não precisa se preocupar com situações excepcionais. Concentre-se na solução em si.