Exercícios

Orientação a Objetos

Básico

1) Construir a classe *Horario* com os seguintes membros (atributos + métodos):
Atributos: *hora, min*
Métodos:

- *getHora()*:
  não recebe valor
  retorna o atributo *hora*
- *getMin()*:
  não recebe valor
  retorna o atributo *min*
- *setHora()*:
  recebe o novo valor da hora
  retorna *true* se a hora for válida, false c. c.
- *setMin()*:
  recebe o novo valor do minuto
  retorna *true* se o min for válido, false c. c.
- *calcularIntervalo()*:
  recebe um horário h (objeto da classe *Horario*) como parâmetro
  calcula o intervalo de tempo deste *Horario* até o *Horario* h
  retorna o valor do intervalo em minutos (int)

Construir uma classe principal para testar a classe *Horario*, a qual:

- pede ao usuário dois horários, de entrada e de saída
- cria dois objetos da classe *Horario*
- calcula o intervalo de tempo entre eles
- calcula quanto custou:
  se intervalo menor do que 3 h -> R$ 4,50
  se intervalo entre 3 h e 12 h -> R$ 0,75 a cada 15 min excedente
  se intervalo maior do que 12 h -> R$ 33,00

2) Adaptar a classe *Horario* do exercício anterior para inicializar os valores de hora e minuto através de uma construtora. Qual(is) é(são) a(s) alternativa(s) para efetuar a validação dentro da construtora e retornar o status (sucesso ou erro)? Esta construtora poderia ser sobrecarregada de alguma forma?

3) Criar uma classe *Caixa*, cujo objetivo é representar uma caixa em um sistema de transporte de cargas.

Atributos:

- largura

- altura

- profundidade

Métodos:

- *setLargura(), setAltura() e setProfundidade()* – permitem ajustar os valores dos atributos ("setters")

- *calcularAreaExt(), calcularVolume()* – calculam e retornam os valores de área externa da caixa e volume, respectivamente.

Criar uma classe ExTestadorDeCaixa que:

- pergunte ao usuário as dimensões de 2 caixas.

- crie 2 objetos da classe Caixa, preencha os atributos (utilizando os "setters") e chame os métodos para mostrar a área externa e o volume de cada uma.

area = 2(largura*altura + largura*profundidade + altura*profundidade)*

*volume = largura*altura*profundidade*

4) *Vide exercícios pertinentes ao grupo de slides 12.*

5) Criar a classe *Pessoa* com as seguintes características:

- atributos: idade e dia, mês e ano de nascimento, nome da pessoa

- métodos:
  - *calculaIdade()*, que recebe como parâmetro a data atual em dia, mês e ano e calcula e armazena no atributo *idade* a idade atual da pessoa, sem retornar valor
  - *getIdade(),* que retorna o valor da idade
  - *getNome(),* que retorna o nome da pessoa
  - *setNome(),* que recebe o nome da pessoa como parâmetro e inicializa o atributo da classe
  - *setDataDeNascimento()*, que recebe dia, mês e ano de nascimento como parâmetros e preenche nos atributos correspondentes do objeto.

- Fazer uma classe principal que crie dois objetos da classe *Pessoa*, um representando Albert Einstein (nascido em 14/3/1879) e o outro representando Isaac Newton (nascido em 4/1/1643). Em seguida, mostre quais seriam as idades de Einstein e Newton caso estivessem vivos.

- Exemplo de classe de entrada (*que está em Java permitindo perceber certa proximidade sintática com C++*):

```
public class ExPessoa
{
      public static void main (String [] args)
      {
            Pessoa p1, p2;

            p1 = new Pessoa();
            p2 = new Pessoa();

            p1.setNome("Isaac Newton");
            p1.setDataDeNascimento(4, 1, 1643);

            p2.setNome("Albert Einstein");
            p2.setDataDeNascimento(14, 3, 1879);

            p1.calculaIdade(30, 9, 2010);
            p2.calculaIdade(30, 9, 2010);

            int idadeP1 = p1.getIdade();
            int idadeP2 = p2.getIdade();

            System.out.println("Idade de p1: " + idadeP1);
            System.out.println("Idade de p2: " + idadeP2);


      }

}
```

6) Alterar o programa do exercício anterior para substituir o método *setDataDeNascimento* e o método *setNome* por uma construtora

7) Implementar a classe *PolReg*, que define um polígono regular

- Atributos: número de lados, tamanho do lado
- Métodos: cálculo do perímetro, cálculo do ângulo interno e cálculo de área. Este último deve retornar o valor zero, dado que não é possível calcular a área de um polígono regular genérico
- Construtora que recebe o número de lados e o tamanho dos lados como parâmetros e inicializa os valores dos atributos

Implementar uma classe principal para testar a classe *PolReg*.

- pedir ao usuário os dados de um polígono regular
- criar o objeto, lembrando de passar os valores para a construtora
- chamar os métodos e mostrar o que eles retornam

8) Implemente uma classe chamada *Carro* com as seguintes propriedades:

- Um veículo tem um certo consumo de combustível (medidos em km/litro), uma certa capacidade máxima de combustível e uma certa quantidade de combustível no tanque.
- O consumo e a capacidade máxima são passados como parâmetro para o construtor e o nível de combustível inicial é 0.
- Forneça um método *andar( )* que simule o ato de dirigir o veículo por uma certa distância, reduzindo o nível de combustível no tanque de gasolina.

- Forneça um método *getCombustivel( )*, que retorna o nível atual de combustível.
- Forneça um método *abastecer( )*, para abastecer o tanque.
- Escreva um pequeno programa em C++ que teste sua classe.
  Obs.: Em necessitando, alguns métodos podem ter parâmetros.
- Eis um exemplo de uso (*só que aqui em Java*):

```
Carro gol = new Carro(12, 45); // 12 quilômetros por litro
                               //de combustível,
                    //capacidade do tanque é 45 litros
gol.abastecer(20); // abastece com 20 litros de combustível.
Carro uno = new Carro(14,40);
uno.abastecer(25);
uno.andar(150); // anda 150 quilômetros.
int sobra = uno.getCombustivel() // Exibe o combustível que resta
no tanque.
System.out.println("Litros restantes no Uno: " + sobra);
gol.andar(80);
sobra = gol.getCombustivel();
System.out.println("Litros restantes no Gol: " + sobra);
```

9) Alterar o exercício anterior para que a classe *Carro* leve em consideração tanto o consumo urbano quanto o consumo em estrada (o método *andar* deve receber um parâmetro que indica se está andando na cidade ou na estrada). Em seguida, implemente um programa que, utilizando (*objeto d*)esta classe, efetue as seguintes operações:

- receba do usuário uma quantidade de km e a quantidade de litros que o carro ~~consumiu~~ TEM para andar aquela quantidade de km
- calcule e mostre quantos km ele andou na estrada e quantos ele andou na cidade.

10) Implementar a classe *Colaborador* com as seguintes características:

- atributos: nome, tempo de serviço na empresa em anos, tipo do vínculo (empregado, sócio ou estagiário), valor da hora de trabalho e número de horas que trabalha.
- métodos:
  - construtora que recebe o nome e o tipo do vínculo do colaborador e zera os demais atributos
  - métodos "setter" para cada um dos outros atributos
  - *calculaRendimentos()*, que calcula e retorna os rendimentos daquele colaborador. As regras para cálculo de rendimento são as seguintes:
    i. Estagiários recebem o valor da hora vezes o número padrão de horas trabalhadas (80 por mês)
    ii. Empregados recebem o número de horas trabalhadas vezes o valor da hora, sendo que este valor aumenta em 10% para cada ano de serviço na empresa. Caso o número de horas trabalhadas exceda 144, o empregado recebe 50% a mais por hora extra.
    iii. Sócios recebem o valor da hora vezes o número de horas trabalhadas.

- *calculaCusto(),* que calcula e retorna quanto aquele colaborador custa mensalmente para a empresa. As regras para cálculo do custo são as seguintes:
    i. Estagiários e sócios não apresentam nenhum custo adicional além dos rendimentos.
    ii. Empregados custam o valor dos seus rendimentos mais 80% relativos a encargos (impostos, INSS etc.).
- *getNome()*, que retorna o nome do colaborador.

Implementar um programa que:

- receba do usuário os dados de 3 colaboradores.
- calcule os rendimentos e custos de cada um e informe qual deles tem o maior rendimento e qual custa mais para a empresa.

11) ~~Implementar a classe *Relogio* em Java, que pretende ser utilizada como modelo para a criação de um relógio com resolução de nanossegundos:~~

- ~~atributos: ano, mês, dia, hora, minutos, segundos e nanossegundos da data/hora ao qual este objeto corresponde.~~
- ~~métodos:~~
    - ~~construtora para iniciar o objeto com os parâmetros fornecidos.~~
    - ~~construtora para iniciar o objeto com a data/hora atuais (dica: utilizar *Calendar.getInstance()*). Como a classe *Calendar* possui resolução de milissegundos, o atributo de nanossegundos receberá um múltiplo de 10^6 (1 ms = 10^6 ns)~~
    - ~~método *getAgora()*, que retorna o valor da hora atual, com precisão de nanossegundos em relação ao valor original. Dica: utilizar *System.nanoSeconds()* para verificar o tempo que se passou desde a criação do relógio, em nanossegundos.~~

~~Implementar um programa para testar esta classe, obtendo os valores de tempo repetidamente e verificando se são valores coerentes.~~

~~*Obs.: Este exercício até poderia ser feito em C++, mas exercitaria mais bibliotecas que POO em si, dado que parte pertinente de POO aí pertinente já foi contemplada em exercício anterior.*~~

12) Fazer uma classe *Vetor* em C++ que pretende representar a entidade geométrica vetor em um espaço bidimensional. Esta classe deve possuir as seguintes características:

- Atributos: *dx* e *dy*
- Operador + sobrecarregado, que recebe outro objeto da classe *Vetor* como parâmetro e retorna um terceiro vetor cujas dimensões são a soma das dimensões dos vetores anteriores.
- Operador * sobrecarregado, que recebe um inteiro como parâmetro e o multiplica pelas dimensões do vetor.
- Construtora que recebe as dimensões como parâmetro e inicializa os atributos do objeto.

Escrever um programa que crie dois vetores e exercite os seus métodos.

13) ~~Repetir o exercício 12) em Java, substituindo os operadores sobrecarregados por métodos que executem as mesmas operações.~~

14) Fazer um programa orientado a objetos para simular a atração gravitacional entre dois corpos celestes:

- o programa instancia um frame, o qual possui uma área de desenho na qual é desenhada uma porção bidimensional do espaço com dimensões em escala planetária (por exemplo, 100e6 km de largura por 100e6 km de altura)
- dois corpos são posicionados aleatoriamente (ou por entrada do usuário) no espaço bidimensional. Os corpos possuem massas em escala planetária (por exemplo, 6e27 kg), também definida pelo usuário
- os corpos possuem velocidades nos eixos x e y, também em escala planetária (por exemplo, 30 km/s) e também definida pelo usuário
- ao iniciar-se o funcionamento do programa, os corpos devem ter sua interação gravitacional calculada (terceira lei de Newton), o que permite calcular a aceleração, variação da velocidade e variação da posição para um intervalo de tempo (também definível pelo usuário). Os corpos devem então ser redesenhados e o processo deve ser repetido, simulando a trajetória dos corpos sujeita à atração gravitacional.
- *Obs.: Este exercício até poderia ser feito em C++, mas exercitaria mais cálculo matemático que POO em si, dado que parte pertinente de POO aí pertinente já foi contemplada em exercício anterior.*