ELE078 - Programação Orientada a Objetos

Atividade Prática 02

Classe Ponto2D

Escreva uma classe Ponto2D em C++ para manipular pontos no espaço cartesiano bidimensional. Os objetos Ponto2D devem ter as coordenadas *x* e *y* do tipo *double*.

Recomendações:

- Use sobrecarga de funções sempre que possível;
- Use lista de inicialização no construtor
- Funções membro que não alteram o estado dos objetos devem ser const.
- Use, sempre que possível, passagem de argumentos por referência implícita, com o modificador const para argumentos cujo valor não deve ser alterado.
- avaliem a possibilidade dos métodos retornarem objetos por cópia ou por referência.
- avaliem a possibilidade de usarem funções inline para otimização do código.

A classe deve possuir:

- construtores que permitem a inicialização de objetos do tipo Ponto2D das seguintes formas:

In [ ]:
```
Ponto2D p1;             // inicializa o ponto2D p1 com as coordenada
s x = 0.0 e y = 0.0;
Ponto2D p2(3.0, 4.0);   // inicializa o ponto2D p2 com as coordenada
s x = 3.0 e y = 4.0;

// inicializações usando o construtor de copia
Ponto2D p3(p1);      // Ponto2D p3 é inicializado com as coordenadas
do ponto p1
Ponto2D p4 = p2;     // Ponto2D p4 é inicializado com as coordenadas
do ponto p2
```

Atributos de Instancia x Atributos Static

Cada objeto Ponto2D deve ter um identificador (*id*) que é gerado de forma automática toda vez que este é inicializado pelo construtor da classe, da seguinte forma: id = getNextId()

Vcs devem criar uma função membro (*private*) que retorna o próximo *id* disponível para o objeto que está sendo inicializado, da seguinte maneira:

int getNextId( ) : gera um novo identificador (*id*) para o objeto, o qual deve ser gerado de forma aleatória entre 0...1000 e, deve ser diferente dos *ids* dos objetos que já foram inicializados e encontram-se atualmente na memória.

Vcs podem criar uma lista estática de *ids* (*static*) para controlar os *ids* que já foram usados durante a execução do programa. No momento em que um objeto sai do escopo e deixa de existir, ele deve liberar seu identificador para que este possa ser eventualmente usado por um novo objeto.

Funções membro *get* and *set* para acessar e alterar os atributos dos objetos:

In [ ]:
```
// Função membro para escrever (imprimir) as coordenadas do ponto na tela
p1.print();
```

In [ ]:
```
// Função membro para calcular a distância do ponto à origem do sistema de coordenadas cartesiano
double d = p1.distToOrig();
```

In [ ]:
```
// Função membro para calcular a distância entre dois pontos p1 e p2, onde p1 é o objeto que chama a função
double d = p1.distTo(p2);
```

In [ ]:
```
// Função membro que modifica as coordenadas do objeto corrente p1 somando às
// suas coordenadas um novo ponto p2, que deve ser passado por argumento.
p1.sumOf(p2);
```

In [ ]:
```
// Função membro que retorna um ponto p3 que é o resultado da soma entre dois pontos p1 e p2,
// onde p1 é o objeto que chama a função.
p3 = p1.sumOfWithReturn(p2);
```

Sobrecarga de Operadores

Implemente as sobrecargas para os operadores de adição (+) e decremento pré-fixado (--) para a classe Ponto2D. Considere a sobrecarga como funções membro de classe conforme código a seguir.

In [ ]:
```
#include<iostream>

class Ponto2D{

    int x,y;

    public:
        Ponto2D(int xx=0, int yy=0):x(xx),y(yy){ };
        Ponto2D& operator-- ();
        Ponto2D operator+ (const Ponto2D& ) const;
        ~Ponto2D(){};
};

int main() {

    Ponto2D a(1,4), b(3,2), c;
    c = a + b;               // soma as coordenadas dos pontos a e b
    --c;                    // decrementa em uma unidade as coordena
das de c
}
```