Linguagem de Programação C++

Uma outra peculiaridade da linguagem C++ é a forma como esta trata a passagem de parâmetro por valor e por referência.

Para nos recordamos de como este processo ocorria na linguagem C analisaremos o programa a seguir que possui uma função responsável por fazer a troca de valores entre duas variáveis.

```
#include <stdio.h>
void Swap (int *a, int *b)
{
        int temp;
        temp = *a;
        *a = *b;
        *b = temp;
}
main ()
{
        int num1, num2;
        printf("\nEntre com um valor inteiro para A: ");
        scanf ("%d", &num1);
        printf("\nEntre com um valor inteiro para B: ");
        scanf ("%d", &num2);
        Swap (&num1, &num2);
        printf ("\nA agora vale %d  e B  vale %d\n",
        num1, num2);
        return 0;
}
```

Linguagem de Programação C++

O programa com a mesma finalidade escrito na linguagem C++ ficará:

```
#include <iostream.h>
using std::cin;
using std::cout;
void Swap (int &a, int &b)
{
        int temp;
        temp=a;
        a=b;
        b=temp;
}
main ()
{
        int num1,num2;
        cout << "Entre com um valor inteiro para A: ";
        cin >> num1;
        cout << "Entre com um valor inteiro para B: ";
        cin >> num2;
        Swap (num1, num2);
        cout << "A agora vale " << num1 << " e B  vale "  << num2;
        return 0;
}
```

Linguagem de Programação C++

Outra singularidade da linguagem C++ é a forma como esta possibilita ao programador manipular a alocação dinâmica de memória.

Para o gerenciamento dinâmico de memória é disponibilizado o operador ***new*** que recebe um tipo de dado e aloca memória para um ou ***n*** elementos do tipo recebido e retorna o endereço de memória da área alocada.

Sua sintaxe é a seguinte:

...

TipoDeDado *p;

p = new TipoDeDado [tamanho];

/* A especificação do tamanho é opcional e sua ausência implica na alocação de área suficiente para armazenar apenas um elemento do tipo especificado.*/

Linguagem de Programação C++

Para uma melhor compreensão, analisaremos o programa a seguir que, lê da entrada padrão o número de linhas e de colunas de uma matriz de números em ponto flutuante, aloca espaço dinamicamente para esta e a inicializa, com valores fornecidos pelo usuário, através da entrada padrão. Ao final o programa retorna a matriz na saída padrão com layout apropriado.

```
#include <iostream.h>
#include <iomanip.h>
using namespace std;
int main ()
{
   int i,j,cont;
   float *matriz;
   cout << "Entre com o numero de linhas da matriz: ";
   cin >> i;
```

```
cout << "Entre com o numero de colunas da matriz: ";
cin >> j;
matriz = new float [i*j];
if (!matriz) {
   cout << endl << "ERRO!" << endl; exit (1); }
for (cont=0; cont<i*j; cont++)
{
   cout << endl << "Entre com o elemento da matriz [" <<
   cont/j+1 << "," << cont%j+1 << "]: ";
   cin >> matriz[cont];
}
for (cont=0; cont<i*j; cont++)
   if (!(cont%j))
      cout << "|" << setw(7) << setprecision (2) <<
      setiosflags (ios::showpoint) << matriz[cont];
   else
      if (cont%j==j-1)
         cout << setw(7) << matriz[cont] << "|" << endl;
      else
         cout << setw(7) << matriz[cont];
return 0;
}
```

Linguagem de Programação C++

Quando se trata de alocação dinâmica de memória, uma operação que torna-se essencial, além da reserva de áreas da memória, é a liberação de áreas reservadas que não serão mais utilizadas.

O operador ***delete*** possibilita a liberação de áreas alocadas, permitindo assim a reutilização das mesmas.

Exemplo:

```
...
char *p_c, *vetor;
p_c = new char;
vetor = new char [10];
delete p_c;
delete []vetor; /* Se usar apenas 'delete vetor', apenas o primeiro elemento do vetor será desalocado. */
```

Linguagem de Programação C++

Exercício:

Com base no que vimos, construa um programa, na linguagem C++, que aloque dinamicamente memória para um vetor de strings. O processamento se dará da seguinte forma: o usuário fornecerá através da entrada padrão um conjunto de strings com tamanhos aleatórios. O final de uma string é identificado pelo pressionamento da tecla ***enter*** e o final do conjunto de strings é identificado pelo fornecimento de uma string vazia pelo usuário. Ao final do processamento o programa deve retornar na saída padrão as strings contidas no vetor.

Linguagem de Programação C++

Neste ponto, já possuímos uma boa visão dos aspectos de C++ que somados aos nossos conhecimentos de C nós possibilitarão explorar adequadamente a orientação a objeto nesta linguagem.

Apenas uma última observação se faz necessária. Objetivando facilitar a compreensão dos aspectos iniciais da linguagem C++ utilizamos até o momento a notação #include <arquivoDeCabecalho.h>, que encontra-se em desuso, sendo considerada uma notação antiquada. Inclusive, alguns compiladores de códigos fonte em C++ não aceitam tal notação. Atualmente se suprime o .h, ou seja, utiliza-se #include <arquivoDeCabecalho>

Linguagem de Programação C++

O primeiro tópico a ser trabalhado versa sobre como definir uma classe, especificando seus atributos, métodos e especificadores de acesso.

Sintaxe para a definição de uma classe:

```
class NomeDaClasse
{
   tipoDoPrimeiroAtributo nomeDoPrimeiroAtributo;
   ...
   tipoDeRetorno nomeDoMetodo (listaDeParametros)
   {
      //corpo do método
   }
   ...
};
```

Observação: Enfatizamos a utilização da grafia camelo (caixa alta e caixa baixa).

Linguagem de Programação C++

Para uma melhor compreensão vamos analisar o exemplo da definição de uma classe denominada Ponto2D, a qual possui dois **membros de dados** (atributos) representando as coordenadas e uma **função membro** (método) para alterar o valor dos atributos.

```
class Ponto2D
{
   float x;
   float y;
   void mover (float novoX, float novoY)
   {
      x = novoX;
      y = novoY;
   }
};
```

Linguagem de Programação C++

Agora veremos como é feita a instanciação de um objeto da classe Ponto2D. O processo se dá da mesma forma que em C é feita a declaração de uma variável de um tipo definido pelo usuário. Por exemplo:

```
...
class Ponto2D
{
   float x;
   float y;
   void mover (float novoX, float novoY)
   {
      x = novoX;
      y = novoY;
   }
};
...
int main()
{
   Ponto2D ponto;
   ...
}
```

Linguagem de Programação C++

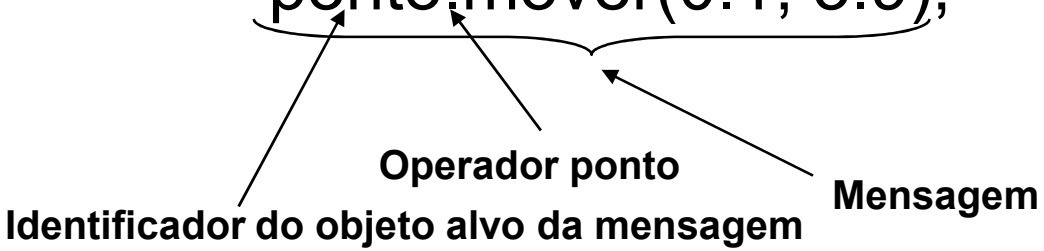
Agora vamos tratar da manipulação de um objeto instanciado.

Como solicitar a um objeto que execute um determinado comportamento?

Enviando uma mensagem ao mesmo.

Em nosso exemplo, para solicitarmos que o objeto ponto execute seu comportamento implementado pela função membro mover enviamos a seguinte mensagem:

```
ponto.mover(0.1, 3.9);
```



Devemos ter em mente que, assim como na linguagem C, os tipos dos argumentos em C++ também devem ser **consistentes**.

Linguagem de Programação C++

Exercício:

Com base no que foi apresentado, efetue testes utilizando a classe Ponto2D.

```
...
class Ponto2D
{
   float x;
   float y;
   void mover (float novoX, float novoY)
   {
      x = novoX;
      y = novoY;
   }
};
...
int main()
{
   Ponto2D ponto;
   ...
}
```

Contudo, a classe Ponto2D que foi definida apresenta um grande problema, gerado pelos especificadores de acesso (visibilidade). Em outras palavras, todos os membros de dados e a função membro foram especificados como privados. Fato que inviabiliza a manipulação de objetos instanciados da classe Ponto2D.

Este problema ocorre devido ao especificador de acesso padrão, assumido por omissão, ser o ***private***, sendo assim, todos os membros depois do cabeçalho de classe e antes do primeiro especificador de acesso são ***private***, ou seja, inacessíveis fora do contexto do objeto.

Como na definição da classe Ponto2D não consta nenhum especificador de acesso, todos os membros são privados.

Este problema é solucionado com a utilização adequada dos especificadores de acesso ***private*** e ***public***.

No caso do nosso exemplo:

```
class Ponto2D
{
  private:
    float x;
    float y;
  public:
    void mover (float novoX, float novoY)
    {
      x = novoX;
      y = novoY;
    }
};
```

Linguagem de Programação C++

Como foi mencionado o especificador de acesso private pode ser suprimido, já que é assumido por omissão. No caso, a definição da classe Ponto2D ficaria:

```
class Ponto2D
{
    float x;
    float y;
  public:
    void mover (float novoX, float novoY)
    {
       x = novoX;
       y = novoY;
    }
};
```

Contudo, esta não é uma prática saudável.

Linguagem de Programação C++

Vamos agora tornar a classe Ponto2D mais robusta, ou seja, possibilitar que o valor dos membros de dados possam ser visualizados através do envio de uma mensagem aos objetos instanciados da referida classe.

Logo, teremos a definição de mais funções membros para retornarem os valores dos membros de dados.

```
class Ponto2D
{
  private:
    float x;
    float y;
  public:
    void mover (float novoX, float novoY)
    {
      x = novoX;
      y = novoY;
    }
```

```
        float getX ()
        {
           return x;
        }
        float getY ()
        {
           return y;
        }
};
```

Uma outra adaptação que pode ser feita é a inserção de duas novas funções membros, excluindo ou não a função membro mover.

Desta forma, a classe Ponto2D ficaria assim:

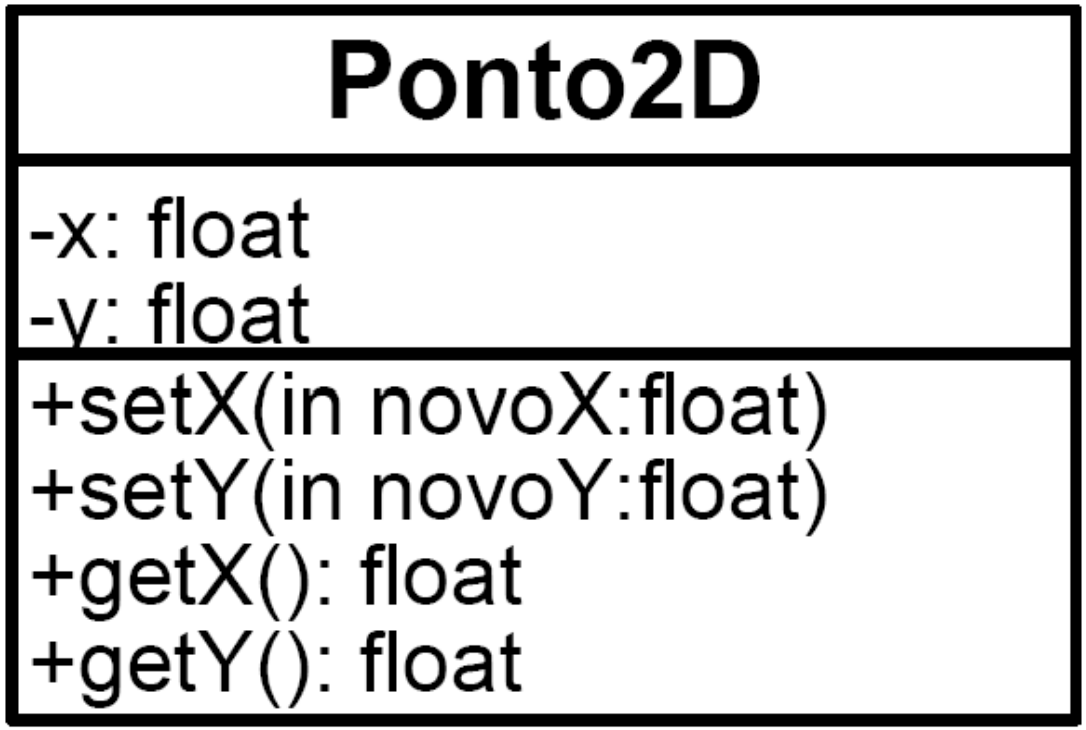
```
class Ponto2D
{
  private:
    float x;
    float y;
  public:
    void setX (float novoX)
    {
      x = novoX;
    }
    void setY (float novoY)
    {
      y = novoY;
    }
    float getX ()
    {
      return x;
    }
    float getY ()
    {
      return y;
    }
};
```

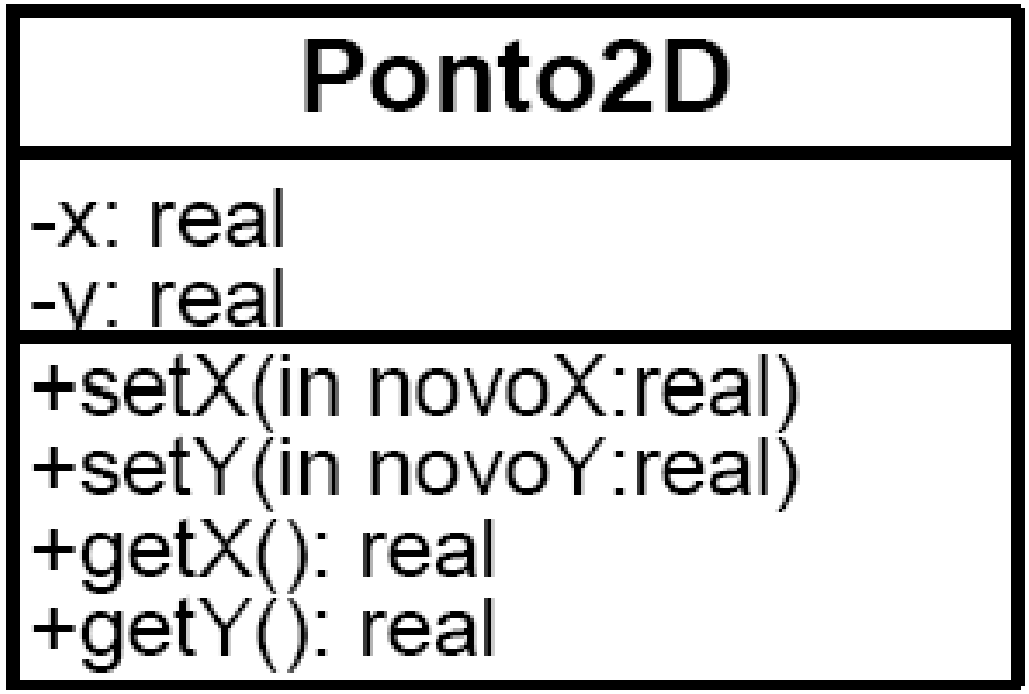
```
class Ponto2D
{
  private:
    float x;
    float y;
    void setX (float novoX)
    {
      x = novoX;
    }
    void setY (float novoY)
    {
      y = novoY;
    }
  public:
    void mover (float novoX, float novoY)
    {
      setX (novoX);
      setY (novoY);
    }
    float getX ()
    {
      return x;
    }
    float getY ()
    {
      return y;
    }
};
```

A nomenclatura **get** e **set** é uma convenção, relevante é salientar que a existência de funções membros desta natureza torna a classe mais robusta. Pois, as manipulações dos atributos podem ser concentradas nestas funções favorecendo a manutenibilidade do código.

Observação: não é necessário fornecer as funções get e set para cada item de dado (atributo) privado; estas capacidades devem ser fornecidas somente quando apropriado.

Exercício: Construa um diagrama de classe UML para representar a classe Ponto2D.


Ponto2D
-x: float
-y: float
+setX(in novoX:float)
+setY(in novoY:float)
+getX(): float
+getY(): float


Ponto2D
-x: real
-y: real
+setX(in novoX:real)
+setY(in novoY:real)
+getX(): real
+getY(): real

Linguagem de Programação C++

O estado de um objeto reflete o conteúdo de seus atributos. Desta forma, seria de grande valia se no momento da instanciação de um objeto existisse uma maneira de determinar seu estado inicial.

Isto é possível, através da utilização de uma função-membro especial denominada **construtor**.

Um construtor deve ser definido com o mesmo nome da classe, possibilitando ao compilador diferenciá-lo, este não pode especificar um tipo de retorno (nem mesmo o void). Normalmente são declarados públicos.

A definição de um construtor possibilitará garantir que um objeto foi inicializado adequadamente antes de sua utilização. Isto ocorre, pois a chamada ao construtor ocorre implicitamente no instante da instanciação de um objeto.

UNIVASF

Linguagem de Programação C++

O compilador da linguagem C++ fornece um construtor padrão, sem parâmetros, para todas as classes definidas.

O construtor padrão não fornece nenhum valor inicial para os membros de dados do objeto, apenas chama o construtor padrão para os membros de dados que são objetos.

Na classe Ponto2D, que foi definida, ficou a cargo do compilador fornecer um construtor padrão.

Um construtor padrão, sem parâmetros, também pode ser definido pelo programador.

Para uma melhor compreensão destes conceitos, vamos definir um construtor padrão para a classe Ponto2D.

UNIVASF

```
class Ponto2D
{
   private:
      float x;
      float y;
   public:
      Ponto2D ()
      {
         x = 0;
         y = 0;
      }
      void setX (float novoX)
      {
         x = novoX;
      }
      void setY (float novoY)
      {
         y = novoY;
      }
      float getX ()
      {
         return x;
      }
      float getY ()
      {
         return y;
      }
};
```

Linguagem de Programação C++

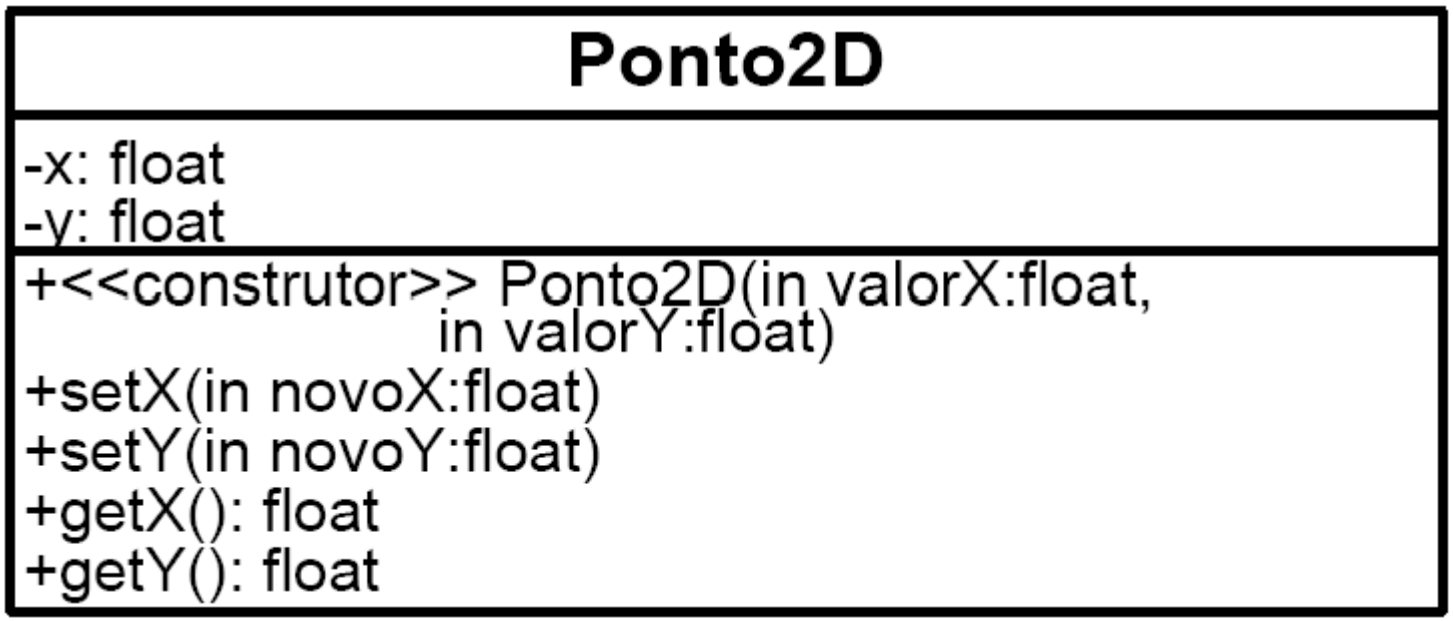
Um construtor também pode ser definido com parâmetros. Por exemplo:

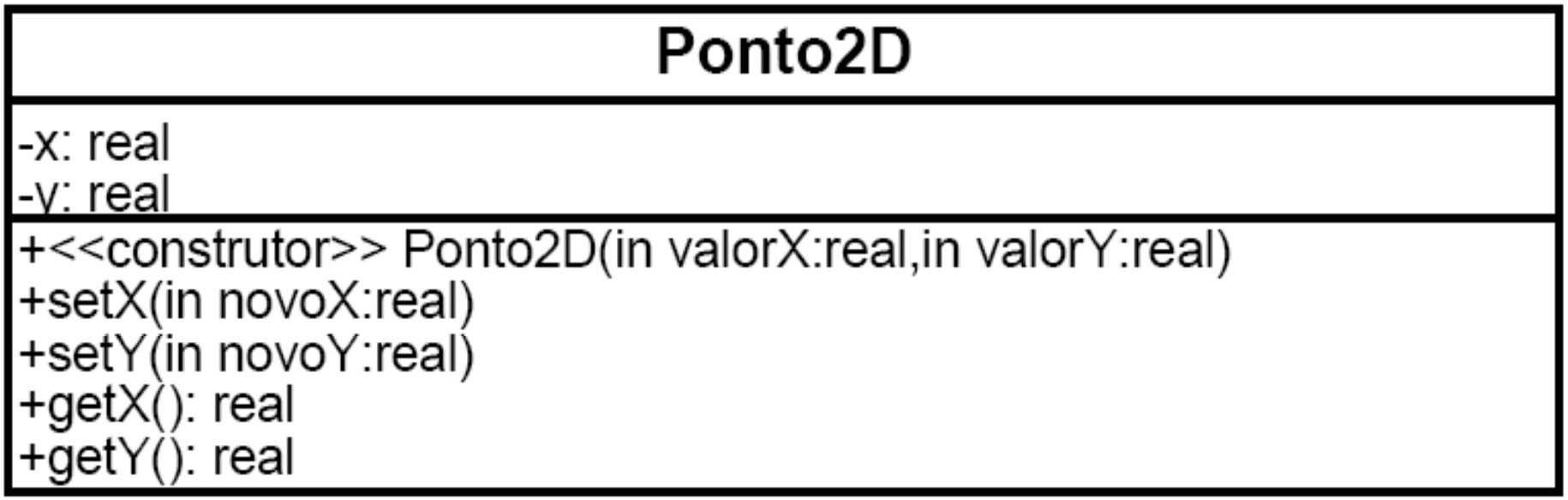
```
class Ponto2D
{
   private:
      float x;
      float y;
   public:
      Ponto2D (float valorX, float valorY)
      {
         x = valorX;
         y = valorY;
      }
      ...
};
```

Observação: Se o programador definir um construtor com parâmetros o compilador não criará um construtor padrão para a classe.

Linguagem de Programação C++

Veremos agora como é representada, em UML, a classe Ponto2D com um construtor.




Ponto2D
-x: real
-y: real
+<<construtor>> Ponto2D(in valorX:real,in valorY:real)
+setX(in novoX:real)
+setY(in novoY:real)
+getX(): real
+getY(): real

Exercício:

Defina uma classe denominada Conta, a qual poderá vir a ser utilizada por um sistema bancário para representar contas de clientes. Dentre os membros de dados que forem especificados deve constar um que seja capaz de armazenar o saldo do cliente ao qual a conta pertence. A classe deve fornecer um construtor capaz de receber um saldo inicial e instanciar o objeto adequadamente. Mais três funções membros devem ser definidas: uma que credite um valor na conta do cliente, outra que debite um valor na conta do cliente e por fim uma que verifique o saldo do cliente. Construa um programa em C++ que se utilize adequadamente da classe definida, testando todas as funções membros definidas.

Exercício:

Construa o diagrama de classes, em UML, para a classe definida no exercício anterior.