Programação Orientada a Objetos

Classes, Objetos e Encapsulamento

Linguagens:

C++

- Compilar em uma IDE ou no terminal usando **g++ <arquivos.cpp> -o output**

  ou **gcc <arquivos.cpp> -lstdc++ -o output**

C#

- Compilar em uma IDE ou no terminal instalando o .NET e usando o comando

  csc -out:output.exe <arquivos.cs>

Atributos e Métodos:

Atributos são variáveis pertencentes a uma classe que armazenam informações sobre os objetos dessa classe. Eles representam as características ou estados dos objetos.

Métodos são funções ou procedimentos definidos em uma classe que descrevem o comportamento dos objetos dessa classe. Eles permitem realizar ações, manipular os atributos e interagir com outros objetos.

Modificadores de Acesso

- Public: O membro declarado como público é acessível por qualquer classe ou instância da classe atual. Ele não impõe restrições de acesso e permite que outros objetos acessem diretamente o membro.

- Protected: O membro declarado como protegido é acessível dentro da classe atual e também pelas subclasses (herdeiras) dessa classe. Ele não é acessível por objetos ou classes que não sejam subclasses diretas da classe em questão.

- Private: O membro declarado como privado é acessível somente dentro da própria classe em que foi declarado. Ele não pode ser acessado por outras classes ou instâncias dessa classe, incluindo subclasses.

Modificador Internal

O modificador "internal" é um nível de acesso em C# que torna um membro (atributo, método, classe) visível apenas dentro do mesmo assembly. Isso significa que o membro marcado como internal só pode ser acessado por classes que estão no mesmo projeto ou assembly. É útil para encapsular e controlar o acesso a determinados membros, restringindo sua visibilidade a um escopo limitado no contexto do assembly.

Modificador Static

O modificador "static" é utilizado para declarar atributos ou métodos em uma classe que pertencem à própria classe, e não a instâncias individuais. Eles podem ser acessados diretamente através do nome da classe, sem a necessidade de criar objetos.

Modificador Virtual

O modificador "virtual" é utilizado para declarar métodos em uma classe base que podem ser sobrescritos por classes derivadas. Isso permite que as classes derivadas forneçam sua própria implementação do método, mantendo a capacidade de se referir ao método da classe base.

Construtores e Destrutores

- Construtores são métodos especiais em uma classe que são usados para inicializar objetos quando são criados. Eles têm o mesmo nome da classe e podem receber parâmetros para definir os valores iniciais dos atributos. Os construtores são chamados automaticamente no momento da criação de um objeto.
- Destrutores, também conhecidos como finalizadores, são utilizados para liberar recursos utilizados por um objeto antes de sua destruição. Eles são invocados automaticamente pelo sistema antes de um objeto ser removido da memória. Os destrutores são declarados usando o til (~) seguido do nome da classe. No entanto, em muitas linguagens, como C#, o uso de destrutores não é comum devido à coleta automática de lixo (garbage collection).

Exemplo de Classe em C++ no mesmo arquivo

```
class Figura
{
private:
    int x, y;
    int tamLado, numLados;

public:
    Figura(int x, int y, int numLados, int tamLado)
    {
        this->x = x;
        this->y = y;
        this->numLados = numLados;
        this->tamLado = tamLado;
    }

    void desenhar()
    {
        printf("Desenhando figura com %d Lados\n", numLados);
    }
};
```

Exemplo de Classe em C++ em arquivos separados

Animal.h

```
class Animal
{
private:
    int numPatas;
    char* nome;

public:
    Animal(int numPatas, char* nome);
    ~Animal();
    void andar();
    void comer();
    void dormir();
    int getNumPatas();
};
```

Animal.c

```cpp
#include "Animal.h"

Animal::Animal(int numPatas, char* nome)
{
    this->numPatas = numPatas;
    this->nome = new char[strlen(nome) + 1];
    strcpy(this->nome, nome);
}

Animal::~Animal()
{
    delete[] nome;
    printf("Destrutor de Animal\n");
}

void Animal::andar()
{
    printf("%s andando com %d patas...\n", nome, numPatas);
}

void Animal::comer()
{
    printf("Comendo...\n");
}

void Animal::dormir()
{
    printf("Dormindo...\n");
}

int Animal::getNumPatas()
{
    return numPatas;
}
```

Main do Programa

```
int main()
{

    Figura *quadrado = new Figura(30, 20, 4, 5);
    Figura *triangulo = new Figura(10, 0, 3, 6);
    Animal *cachorro = new Animal(4, (char*) "Magrelinho");

    quadrado->desenhar();
    triangulo->desenhar();
    cachorro->andar();

    return 0;
}
```

Exemplo de Classe em C#

```
namespace Figuras
{
    public class Figura
    {
        private double x;
        private double y;
        private double numLados;
        private double tamLado;

        public Figura(double x, double y, double numLados, double tamLado)
        {
            this.x = x;
            this.y = y;
            this.numLados = numLados;
            this.tamLado = tamLado;
        }

        public double Perimetro()
        {
            return numLados * tamLado;
        }

        public void Desenhar()
        {
            Console.WriteLine("Desenhando figura com {0} lados", numLados);
        }
    }
}
```

Main do Programa

```
using System;
using Figuras;

namespace Program
{
    class MainClass
    {
        public static void Main(string[] args)
        {
            Figura f = new Figura(0, 0, 4, 10);
            f.Desenhar();
            Console.WriteLine("Perimetro: {0}", f.Perimetro());
        }
    }
}
```