

IMD0030

LINGUAGEM DE PROGRAMAÇÃO I

Aula 06 – Introdução à Orientação a Objetos: Classes e Objetos

Objetivos desta aula

- Introduzir o paradigma de Programação Orientada a Objetos (POO)
 - Conceitos fundamentais: classes e objetos
- Para isso, estudaremos:
 - Como o usuário pode criar tipos por meio de classes
 - Como manipular tipos criados utilizando a linguagem C++
- Ao final da aula, espera-se que o aluno seja capaz de:
 - Entender os conceitos básicos de POO
 - Implementar programas utilizando classes e objetos em C++

Contextualização

- Tipos estrutura (*structs*) representam uma maneira através da qual um usuário podem criar seus próprios tipos nas linguagens C e C++ a partir de tipos já existentes ou outros por ele criados
 - Estruturas são também chamadas de **tipos compostos**
- Exemplo: modelagem de um retângulo e funções para manipular variáveis desse tipo

```
struct Retangulo {  
    int largura;  
    int altura;  
};
```

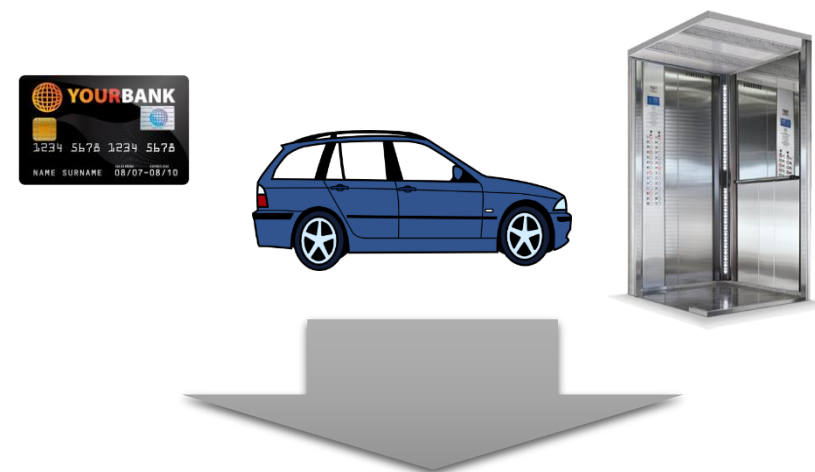
```
int area(struct Retangulo r) {  
    return r.largura * r.altura;  
}  
  
int perimetro(struct Retangulo r) {  
    return (2 * r.largura + 2 * r.altura);  
}
```

Programação Orientada a Objetos (POO)

- **Paradigma de programação** que surgiu na década de 1980 com o objetivo principal de **facilitar o desenvolvimento de programas**
 - aproveitando as melhores ideias da programação estruturada (1960-)
 - agregando novos conceitos para a representação do mundo real de forma mais intuitiva
 - procurando melhorar produtividade e qualidade
- A solução de problemas utilizando POO é baseada na **abstração**
 - das **entidades** do mundo real a serem representadas no programa
 - dos **dados** associados a tais entidades
 - das **operações** que podem ser realizadas por tais entidades

Abstração

- É um dos conceitos fundamentais em POO
- Utiliza-se a técnica de abstração para mapear entidades do mundo real, dando origem ao modelo de classes a ser implementado
- Toma-se como **entidade** tudo o que é real, tendo em consideração as suas características e ações
 - De forma geral, cada **entidade** será mapeada como **classe**
 - As **características** serão mapeadas como **atributos** da classe
 - As **ações** serão mapeadas como **métodos** da classe



| Entidade | Características | Ações |
|-------------------|---|--------------------------------|
| Carro | Cor, peso, modelo, ano, placa | Ligar, desligar, mover, freiar |
| Elevador | Tamanho, capacidade em pessoas, peso máximo | Subir, descer, escolher andar |
| Cartão de Crédito | Titular, numero, saldo, senha, limite | Debitar, sacar, gerar extrato |

Abstração

- Através de um modelo abstrato, pode-se concentrar nas características relevantes e ignorar as irrelevantes
- **Abstração é fruto do raciocínio**
- Através da abstração é possível controlar a complexidade

Encapsulamento

- Esconder os detalhes da implementação de um objeto é chamado **encapsulamento**
 - Conceito que indica que os dados contidos em um objeto **somente** poderão ser acessados e/ou modificados através de seus métodos
 - Não se deve permitir acesso direto aos atributos de uma classe
 - Assegura que toda a comunicação com o objeto seja realizada por um conjunto pré-definido de operações
- Benefícios
 - O código cliente pode usar apenas a interface para a operação
 - A implementação do objeto pode mudar, por exemplo, para corrigir erros ou aumentar desempenho, sem que seja necessário modificar o código do cliente
 - A manutenção é mais fácil e menos custosa
 - Cria um programa legível e bem estruturado

Desenvolvimento Orientado a Objetos

Análise Orientada a Objetos

É o processo de construção de modelos do domínio do problema, identificando e especificando um conjunto de objetos que interagem e comportam-se conforme os requisitos estabelecidos para o sistema.

Projeto Orientado a Objetos

É o processo de geração de uma especificação detalhada do software a ser desenvolvido, de tal forma que esta especificação possa levar a direta implementação no ambiente alvo.

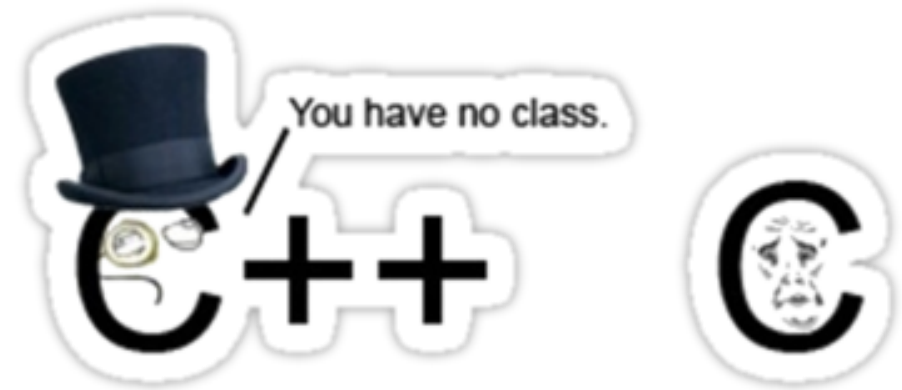
Programação Orientada a Objetos

É um modelo de programação que baseia-se em conceitos como classes, objetos, herança, etc. Seu objetivo é a resolução de problemas baseada na identificação de objetos e o processamento requerido por estes objetos, e então na interação entre estes objetos.

Classes e objetos

Classe: mecanismo provido pelo C++ (e por outras linguagens de programação) para criar tipos compostos

- Apresenta similaridades com os tipos estrutura, porém representa **um conceito novo e mais genérico**
- A introdução de POO na linguagem C++ representa um dos maiores saltos em comparação à linguagem C



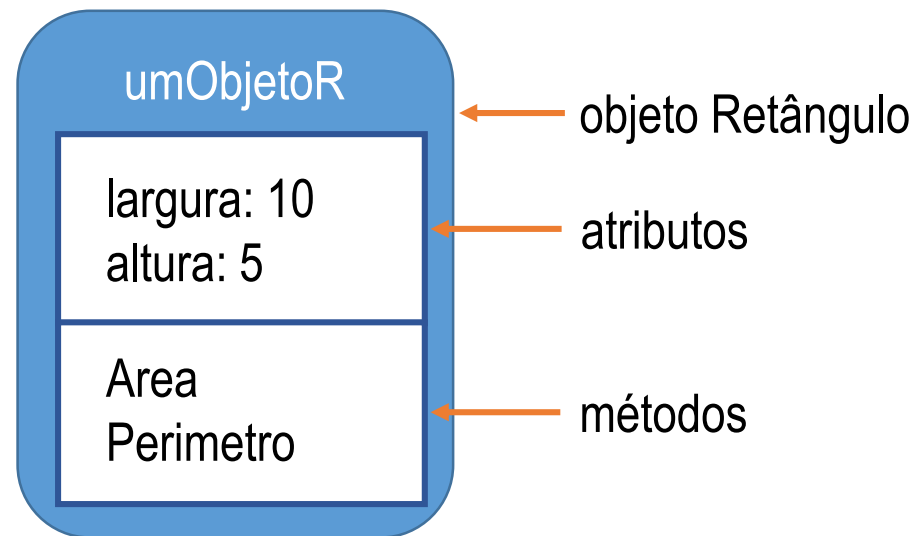
Classes e objetos

Objeto: instância de uma classe

- **Instanciar** um objeto = **criar** um objeto a partir de sua definição (ou seja, de sua classe)
- Um objeto pode ser entendido como uma variável em memória autocontida, responsável pelo seu conteúdo definido na classe
 - Uma classe pode ser também entendida como o tipo de um objeto
- Um objeto tem uma **identidade**, um **estado** e um **comportamento**
 - Identidade: endereço do objeto em memória
 - Estado: valores dos dados armazenados pelo objeto
 - Comportamento: descrito por meio de operações que manipulam os dados gerenciados pelo objeto

Classes e objetos

- Classes definem **membros para um objeto**
 - **Atributos (*data members*)**: informações a serem armazenadas por um objeto
 - **Métodos (*function members*)**: funções que podem ser chamadas por um objeto e manipulam suas informações



Criando classes em C++

- Utilização da palavra-chave **class**, seguida do nome a ser dado à classe
- A definição (corpo) da classe, delimitada por chaves, pode conter
 - uma sequência de **atributos**, declarados de maneira similar a uma variável, e
 - uma sequência de **métodos**, declarados de maneira similar a uma função (métodos *inline*)
- Atributos podem ser acessados por qualquer método da classe

Criando classes em C++ (1)

Exemplo: modelagem de um retângulo

```
class Retangulo {  
    int largura;  
    int altura;  
  
    int area() {  
        return largura * altura;  
    }  
  
    int perimetro() {  
        return (2 * largura + 2 * altura);  
    }  
};
```

} atributos

} métodos (*inline*)

Criando classes em C++

- A implementação dos métodos de uma classe pode ser feita **em separado** à sua definição (interface), o que é uma **boa prática de programação**
 - Maior modularidade, uma vez que basta fazer a inclusão apenas do cabeçalho que define a interface
 - Melhor legibilidade, principalmente para métodos mais longos
 - Conhecimento apenas da interface, sem revelar detalhes de sua implementação
- Uso do **operador de resolução de escopo** `::` para fazer referência a um método de uma classe, quando sua implementação estiver sendo feita fora dela
 - Sintaxe: `<nome da classe> :: <nome do método>`

Criando classes em C++ (2)

Exemplo: modelagem de um retângulo

retangulo.h

```
class Retangulo {  
    int largura;  
    int altura;  
  
    int area();  
    int perimetro();  
};
```

retangulo.cpp

```
#include "retangulo.h"  
  
int Retangulo::area() {  
    return largura * altura;  
}  
  
int Retangulo::perimetro() {  
    return (2 * largura + 2 * altura);  
}
```

Criando objetos em C++

- A **criação de um objeto** de uma classe é praticamente idêntica à declaração de uma variável em C++
- Exemplo: criação de um objeto da classe `Retangulo`

```
int main() {  
    Retangulo r;  
  
    return 0;  
}
```

Criando objetos em C++

- A **criação de um objeto** de uma classe pode também ser feita **de forma dinâmica**, de maneira similar à criação de ponteiros
 - Útil quando algum atributo da classe representa um ponteiro a ser alocado dinamicamente
 - Uso do operador de alocação dinâmica em memória **new**
- Exemplo: criação dinâmica de um objeto da classe `Retangulo`

```
int main() {  
    Retangulo* r = new Retangulo();  
  
    return 0;  
}
```

Visibilidade

- A **visibilidade** de um membro (atributo ou método) de uma classe define a partir de onde ele pode ser acessado
- Três níveis de visibilidade
 - **Pública (*public*)**: acesso direto a partir de qualquer lugar, seja dentro ou fora da classe
 - **Privada (*private*)**: acesso exclusivo a métodos da própria classe
 - **Protegida (*protected*)**: acesso apenas por métodos da própria classe ou de suas sub-classes
- **Por padrão, todo e qualquer membro de uma classe é privado**
 - É possível especificar diferentes visibilidades respectivamente por meio das palavras-chave **public**, **private** e **protected**, seguidas de **:** (dois-pontos)

Criando classes em C++ (3)

Exemplo: modelagem de um retângulo

```
class Retangulo {  
    public:  
        int largura;  
        int altura;  
        int area();  
        int perimetro();  
};
```

Visibilidade

- O **acesso a atributos públicos** de uma classe pode ser feito por meio do **operador de acesso** . (ponto), de maneira simular ao acesso em um tipo estrutura
 - Exemplo: `umRetanguloR.largura = 5`
- O **acesso a métodos públicos** de uma classe também é feito por meio do **operador de acesso** . (ponto)
 - Exemplo: `umRetanguloR.area()`
 - Métodos privados só podem ser chamados por outros métodos públicos da mesma classe → geralmente são métodos auxiliares
- Se um objeto foi criado de maneira dinâmica, o operador de acesso a ser utilizado é o mesmo usado para acessar campos de um ponteiro para estrutura, ->
 - Exemplo: `ptrRetanguloR->area()`

Criando classes em C++ (4)

Exemplo: modelagem de um retângulo

retangulo.h

```
class Retangulo {  
    public:  
        int largura;  
        int altura;  
  
        int area();  
        int perimetro();  
};
```

main.cpp

```
#include "retangulo.h"  
  
int main() {  
    Retangulo r;  
    r.largura = 10;  
    r.altura = 5;  
  
    return 0;  
}
```

Criando classes em C++ (5)

Exemplo: modelagem de um retângulo

retangulo.h

```
class Retangulo {  
    public:  
        int largura;  
        int altura;  
  
        int area();  
        int perimetro();  
};
```

main.cpp

```
#include "retangulo.h"  
  
int main() {  
    Retangulo* r = new Retangulo; ← alocação  
    r->largura = 10;                dinâmica  
    r->altura = 5;  
  
    return 0;  
}
```

Visibilidade

- **Não é possível acessar atributos privados de uma classe por meio do operador de acesso . (ponto)**
 - Conceito de **encapsulamento**: restringir o acesso aos atributos da classe
 - Encapsulamento é a técnica que faz com que detalhes internos do funcionamento dos métodos de uma classe permaneçam ocultos para os objetos
 - Com isso, o conhecimento a respeito da implementação interna da classe é desnecessário do ponto de vista do objeto, visto que isso passa a ser responsabilidade dos métodos da classe
 - Vantagens:
 - Boa prática de programação
 - Eliminar a necessidade de conhecer a estrutura interna da classe
 - Evitar possíveis inicializações e modificações ilegais, o que poderia acontecer com os atributos públicos (e com variáveis de tipo estrutura)

Visibilidade

- Para acessar atributos privados de uma classe, é necessário **criar métodos públicos que os acessem** (métodos *accessors*)
 - **Métodos *getters***: retornam o valor do atributo
 - Convenção: <tipo de retorno> **get**<nome do atributo>()
 - **Métodos *setters***: modificam o valor do atributo atribuindo outro valor recebido como parâmetro
 - Convenção: **void** **set**<nome do atributo>(<parâmetro>)

Criando classes em C++ (6)

Exemplo: modelagem de um retângulo

retangulo.h

```
class Retangulo {  
    private:  
        int largura;  
        int altura;  
  
    public:  
        int getLargura();  
        void setLargura(int l);  
        int getAltura();  
        void setAltura(int a);  
};
```

retangulo.cpp

```
#include "retangulo.h"  
  
int Retangulo::getLargura() {  
    return largura;  
}  
  
void Retangulo::setLargura(int l) {  
    largura = l;  
}
```

Criando classes em C++ (6)

Exemplo: modelagem de um retângulo

main.cpp

```
#include <iostream>
#include "retangulo.h"

int main() {
    Retangulo r;
    r.setLargura(10);
    r.setAltura(5);
    std::cout << "Largura = " << r.getLargura() << ", Altura = " << r.getAltura();

    return 0;
}
```

Exercitando...

1. Implemente uma classe que represente um dado (de jogos de tabuleiro) comum. Crie um programa para instanciar e testar **N** lançamentos de um dado. O valor de **N** deve ser lido da linha de comando.
2. Implemente outro programa para permitir o lançamento de **M** dados e imprimir a soma obtida. O valor de **M** deve ser lido da linha de comando.

Alguma Questão?

