A02 Classes e objetos

📕 Classe

Definição de uma classe:

```
class Retangulo {
public:
    Retangulo();
    Retangulo(int lado);
    Retangulo(int base, int altura);
    ~Retangulo();

    int calcularArea();
    int calcularPerimetro();
    void desenhar();

private:
    int _base;
    int _altura;
};
```

Membros públicos e privados

Os membros definidos como `public` são visíveis por qualquer função da classe ou por qualquer outra função do programa. Nesta seção são declaradas as funções que fazem parte da interface com a qual outros objetos podem interagir[1].

Os membros definidos como `private` estão visíveis apenas para as funções da própria classe. Um membro privado não pode ser acessados por outros objetos ou funções do programa[1].

Atributos

Os **atributos**, ou variáveis de instância, são variáveis que podem ser acessadas por qualquer função da classe. São utilizado para representar e descrever as características dos objetos[1].

Os atributos devem ser sempre privados para garantir o **encapsulamento**. Uma convenção é prefixar os nomes dos atributos com `_` para indicar que são privados e evitar problema de conflitos de nomes[1].

```
private:
    int _base;
    int _altura;
```

Métodos

Os **métodos** são funções declaradas dentro de uma classe. Os métodos podem ser dos seguintes tipos[1]:

- Construtores
- Destrutores
- Acessores e mutantes

Construtores

Os construtores são métodos especiais que possuem o mesmo nome da classe e não possuem valor de retorno. A sua finalidade é inicializar os valores dos atributos, sendo sempre chamados na instanciação dos objetos. Se nenhum construtor for definido, o compilador

```
Retangulo::Retangulo() {
    _base = 0;
    _altura = 0;
}
```

```
Retangulo::Retangulo(int lado) {
    _base = lado;
    _altura = lado;
}

Retangulo::Retangulo(int base, int altura) {
    _base = base;
    _altura = altura;
}
```

O construtor pode ser sobrecarregado para receber diferentes tipos de parâmetros. A decisão para definição dos construtores é uma definição de projeto[2].

O compilador cria um construtor padrão (sem implementação e sem parâmetros) caso a classe não tenha nenhum construtor definido. O compilador também sempre cria um construtor de cópia para toda classe. O construtor de cópia recebe apenas um parâmetro que é um objeto do mesmo tipo da classe. Este construtor pode ser sobrescrito[2].

Outra maneira de inicializar os atributos é usando a notação `: _atributo(valor)`. Exemplo de uso desta notação[2]:

```
Retangulo::Retangulo() : _base(0), _altura(0) {
}

Retangulo::Retangulo(int lado) : _base(lado), _altura(lado) {
}

Retangulo::Retangulo(int base, int altura) : _base(base), _altu-
ra(altura) {
}
```

Essa notação é usada para evitar que parâmetros do tipo objeto sejam instanciados de maneira desnecessária e prevenir que ocorram erros de vazamento de memória[2]. Por exemplo:

```
Retangulo::Retangulo(Ponto p1, Ponto p2) {
    _p1 = p1;
    _p2 = p2;
}
...
Ponto x1(10, 10);
Ponto x2(20, 20);

Retangulo r(x1, x2);
```

Na chamada da criação do objeto, o construtor da classe `Retangulo` instancia os parâmetros `p1` e `p2`, chamando os seus construtores padrão. Em seguida, o compilado copia os valores passados como parâmetro, `x1` e `x2`, e copia para as variáveis do parâmetro, `p1` e `p2`. Com isso, a primeira chamada dos construtores das variáveis do parâmetro são desnecessárias. Caso o construtor desta classe tivesse alocado memória dinamicamente, depois da atribuição, os valores originais seriam perdidos e a memória alocada estaria perdida[2].

Destrutor

O destrutor é um método especial que possui o mesmo nome da classe, prefixado com `~`, não recebe parâmetros e não possui valor de retorno. O destrutor é chamado toda vez que o objeto é removido da memória. Isso ocorre quando uma variável é desalocada explicitamente ou o escopo da variável termina[1].

```
Retangulo::~Retangulo() {
}
```

O destrutor tem como finalidade liberar quaisquer recursos (memória, rede, etc.) que tenham sido alocados pelo objeto. O compilador cria um destrutor vazio (sem implementação) caso a classe não tenha nenhum construtor definido[1].

Acessores e mutantes

A finalidade destes métodos é garantir o acesso dos atributos por outros objetos sem que eles acessem diretamente. Assim, é possível garantir que o objeto armazene apenas valores válidos de acordo com suas regras de comportamento. Por exemplo, uma classe Relógio pode definir como regra que o valor de uma hora deve estar no intervalo entre 0 e 23[2].

Os métodos acessores, ou *getters*, são usados para recuperar os valores dos atributos da classe. Normalmente, os acessores são prefixados com a palavra `get`. Os métodos acessores não precisam estar associados diretamente a uma variável de atributo. Por exemplo, uma classe Relógio pode possuir o método `getSegundos()` que calcula o tempo em segundos com base na hora e nos minutos armazenados[2].

Os métodos mutantes, ou *setters*, são usados para alterar os valores dos atributos da classe. Normalmente, os mutantes são prefixados com a palavra `set`[2].

Outros métodos

Uma classe pode implementar outros tipos de métodos que podem por exemplo iniciar outras tarefas ou realizar cálculos. O nome destes métodos deve sempre começar com um verbo para indicar a ação que ele realizar. Uma prática é usar o verbo sempre no infinitivo para manter a padronização de todo o código do projeto.

Além disso, um método deve sempre realizar somente uma ação e bem feita. Caso um método esteja realizando mais de uma ação, ele deve ser quebrado em outros métodos menores. Um indicativo de que um método está executando mais ações do que deveria é o número de linhas de código implementadas. O ideal é que o código de um método caiba inteiro na tela (cerca de 30~35 linhas).

📙 Objetos

Os objetos são criados a partir de classes. O construtor da classe é sempre invocada durante a instanciação do objeto[1].

```
Retangulo r1;       // chama o construtor Retangulo()
Retangulo r2(3);    // chama o construtor Retangulo(int)
Retangulo r3(3, 4); // chama o construtor Retangulo(int, int)

Retangulo r4 = 10;  // chama o construtor Retangulo(int)

Retangulo r5(r2);   // chama o construtor Retangulo(Retangulo)
Retangulo r6 = r1;  // chama o construtor Retangulo(Retangulo)

Retangulo *r7 = new Retangulo();
Retangulo *r8 = new Retangulo(3, 4);
```

Uma vez criado um objeto, ele pode interagir com os demais objetos do sistema e executar as suas ações. Por exemplo:

```
r2.calcularArea();
```

👨‍💻 Atividade prática

1. Escreva uma classe que represente o modelo de dados de um relógio. O relógio deve conter os atributos de hora e minuto. As seguintes operações devem ser definidas:
    - Inicialização do relógio
    - Definir as horas (0..23)
    - Definir os minutos (0..59)
    - Imprimir o horário (imprimir no formato `hh:mm`)
    - Calcular o número de minutos do horário

📝 Atividade teórica

1. Explique qual é a finalidade de uma classe a sua relação com objetos.
2. Qual é a finalidade de um construtor? E qual a finalidade de um destrutor?
3. O que o compilador faz quando omitimos a declaração de um construtor ou destrutor?
4. Em quais situações podemos omitir a declaração de um construtor?
5. Considere a seguinte afirmativa e justifique se ela é verdadeira ou falsa: *"Em um sistema orientado a objetos, podemos ter classes sem atributos"*
6. Considere a seguinte afirmativa e justifique se ela é verdadeira ou falsa: *"Em um sistema orientado a objetos, podemos ter classes sem métodos."*
7. Qual é a finalidade dos métodos acessores e mutantes?
8. Quais são as desvantagens do uso de métodos acessores e mutantes?

📖 Leitura recomendada

- **Capítulo 8**: MIZRAHI, Vctorine Viviane. **Treinamento em Linguagem C++ - Módulo II**. Makron Books, 1994.[1]

📚 Referência bibliográficas

[1] MIZRAHI, Vctorine Viviane. **Treinamento em Linguagem C++ - Módulo II**. Makron Books,1994. ↩

[2] STROUSTRUP, Bjarne; LISBÔA, Maria Lúcia Blanck; LISBÔA, Carlos Arthur Lang (Trad.). **A linguagem de programação C++**. 3 ed. Porto Alegre, RS: Bookman, 2000. ↩