Introdução a Técnicas de Programação

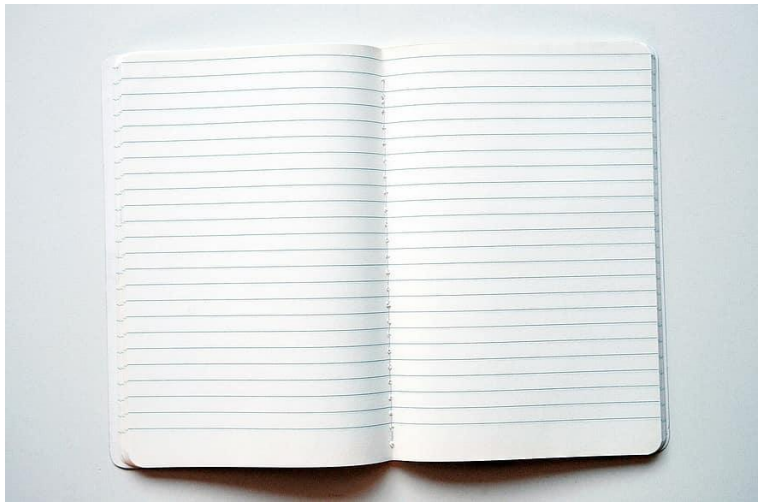
Introdução a Classes

Prof. André Campos
DIMAp/UFRN

Dos dados estruturados aos objetos

Conceito básico de “objetos”: elementos que possuem

- Dados
- Operações sobre seus dados



Caderno → Bloco de notas

- Dados
  - Páginas
  - Textos nas páginas
- Operações sobre os dados
  - Escrever
  - Apagar algo escrito
  - Buscar um texto
  - ...

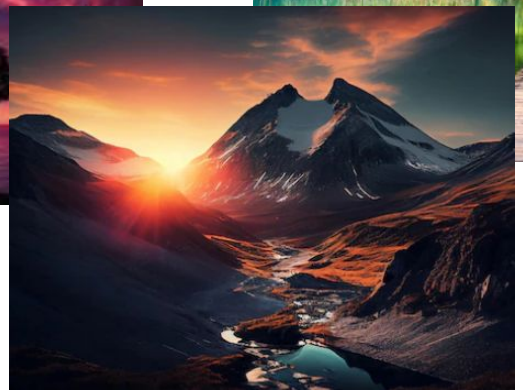
Exemplo

Imagem

- Dados
  - Largura e altura
  - Matriz de pixels
- Operações sobre os dados
  - Rotacionar
  - Ampliar
  - Mudar escala de cores (ex: cinza)
  - ...

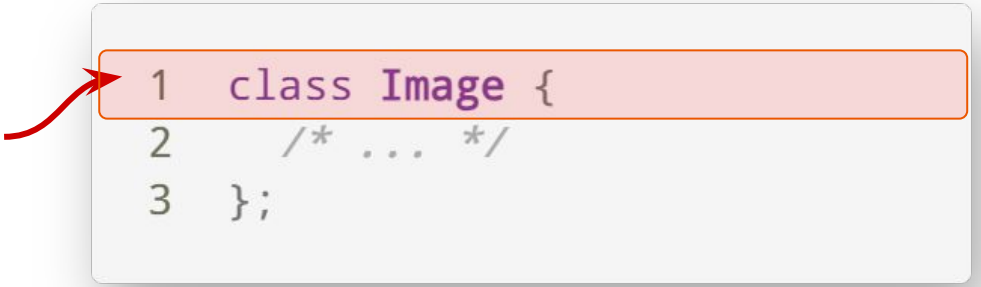
Classes

“Template” (modelo) para a criação de objetos (instâncias) com as mesmas características



Classes em C++

Palavra reservada para definir uma classe e o nome da classe



Classes em C++

Dados (atributos)

```
1  class Image {
2    int width, height;
3    Pixel pixels[256][256];
4
5    /* ... */
6  };
```

Classes em C++

```
1  class Image {
2    int width, height;
3    Pixel pixels[256][256];
4
5    void read_ppm() {/*...*/}
6    void save_ppm() {/*...*/}
7    void rotate() {/*...*/}
8  };
```

Operações (métodos)

Atributos e métodos são chamados a partir de uma instância (objeto)

Instâncias de classes

Uma classe define também um **novo tipo de dado**

Uma variável do tipo da classe é uma **instância** da classes (objeto)

Cada instância tem seus próprios dados (width, height, pixels...)

```
1  class Image {
2    int width, height;
3    Pixel pixels[256][256];
4
5    void read_ppm() {/*...*/}
6    void save_ppm() {/*...*/}
7    void rotate() {/*...*/}
8  };
9
10 int main() {
11   Image original;
12   Image escala_de_cinza;
13   //...
14 }
```

Encapsulamento

Por padrão, atributos e métodos pertencem exclusivamente ao objeto.
Não são visíveis em outra parte do código, exceto dentro da própria classe.
Só o que é definido como **público** é possível alterar.

Público

- Botões de liga/desliga
- Botões de volume
- Touch-screen
- ...

Privado

- Placa principal
- Circuitos
- Transformador de energia
- ...

Acesso público e privado

Por padrão, atributos e métodos são privados.

A partir da palavra reservada **`public`**, eles se tornam públicos.

Se necessário definir novos dados privados, usa-se a palavra reservada **`private`**

```
1  class Image {
2    int width, height;
3    Pixel pixels[256][256];
4    public:
5      void read_ppm() {/*...*/}
6      void save_ppm() {/*...*/}
7      void rotate() {/*...*/}
8  };
9
10 int main() {
11   Image imagem;
12   imagem.read_ppm();    // ok!
13   imagem.rotate();      // ok!
14   imagem.width = 100;   // erro!
15   //...
16 }
17
```

Acesso a dados privados

Se necessário, podemos tornar público o acesso a dados privados através de um método público.

Outras partes do programa não devem alterar as dimensões da imagem, mas podem consultá-las.

```
1  class Image {
2    int width, height;
3    Pixel pixels[256][256];
4
5    public:
6      int get_width() { return width; }
7      int get_height() { return height; }
8      void read_ppm() { /*...*/ }
9    //...
10 };
11
12 int main() {
13   Image imagem;
14
15   imagem.read_ppm();
16   cout << imagem.get_width() << " por ";
17   cout << imagem.get_height() << endl;
18   //...
19 }
```

Construtores e destrutores

Métodos especiais para:

- **Construtor**: inicializar atributos classe, acessar recursos (ex: arquivos)...
- **Destrutor**: liberar recursos (ex: fechar arquivos, liberar memória alocada), ...

O construtor é chamado logo após uma região da memória for atribuída à instância da classe (objeto) que está sendo criada. Seus dados precisam ser inicializados.

O **destrutor é chamado logo antes** da memória atribuída a uma instância ser liberada. Eventuais recursos podem ser liberados.

Construtor

```
1  class Image {
2    int width, height;
3
4    public:
5      Image() { /* ... */ }
6    //...
7  };
```

Método especial:

- Possui o mesmo nome da classe;
- Não possui tipo de retorno.
  - O compilador sabe que o tipo de retorno é a própria classe
- O construtor precisa ser “público” para que um objeto da classe possa ser criado em outras partes do código.

Principais tipos de construtor:

- Construtor padrão (default)
- Construtor parametrizado (com parâmetros)
- Construtor de cópia

Construtor padrão

É chamado quando declaramos uma variável do tipo da classe.

Não há parâmetros!

Os atributos do objeto precisam ser inicializados com um valor padrão (*default*)

Obs: se não for necessário inicializar os atributos, o compilador cria um construtor default automaticamente.

```
1   class Image {
2     int width, height;
3
4     public:
5       Image() {
6         width = 0;
7         height = 0;
8       }
9     //...
10  };
11
12  int main() {
13    Image imagem;
14    //...
15  }
```

Construtor parametrizado

É chamado quando declaramos uma variável passando **parâmetros** para a inicialização.

Os parâmetros são definidos como em um método qualquer (tipo e nome do parâmetro)

```
1   class Image {
2     int width, height;
3
4     public:
5       Image(int w, int h) {
6         width = w;
7         height = h;
8       }
9     //...
10  };
11
12  int main() {
13    Image imagem(100, 100);
14    //...
15  }
```

Construtor parametrizado

É possível especificar quais atributos serão inicializados a partir de quais parâmetros através de uma sintaxe especial

Nessa sintaxe, podemos separar a inicialização de dados de um eventual processo (lógica) a ser realizada com eles.

```
1  class Image {
2    int width, height;
3
4    public:
5      Image(int w, int h): width(w), height(h) {
6        // ...
7      }
8    //...
9  };
10
11 int main() {
12   Image imagem(100, 100);
13   //...
14 }
```

Construtor default parametrizado

Da mesma forma que parâmetros de funções e métodos podem ter valores padrões, os construtores também.

Nesse caso, pode-se inicializar objetos com ou sem parâmetros.

```
1  class Image {
2    int width, height;
3
4    public:
5      Image(int w = 0, int h = 0): width(w), height(h) {
6        // ...
7      }
8    //...
9  };
10
11 int main() {
12   Image imagem_a;
13   Image imagem_b(10, 20);
14   //...
15 }
16
```

Construtor de cópia

É chamado quando declaramos uma variável **passando outro objeto do mesmo tipo** para a inicialização.

Passa a referência constante do objeto que você quer copiar e depois pode acessar seus dados para fazer a cópia.

```
1  class Image {
2    int width, height;
3
4    public:
5      Image(int w = 0, int h = 0): width(w), height(h) {
6        //...
7      }
8
9      Image(const Image& img) {
10       width = img.width;
11       height = img.height;
12       // ...
13     }
14   //...
15 };
16
17 int main() {
18   Image imagem_a;
19   Image imagem_b(imagem_a);
20   //...
21 }
```

Introdução a Técnicas de Programação

Introdução a Classes

Prof. André Campos
DIMAp/UFRN

Práticas

1. Crie uma classe **Contato** para representar os dados de contato de uma pessoa, com os atributos privados de nome, telefone e email. Defina os métodos públicos necessários para consultar e alterar os dados privados de uma pessoa.
2. Crie uma classe **Contatos** que armazena até 10 pessoas e seja capaz de operações como:
    a. Inserir uma pessoa nos contatos.
    b. Buscar uma pessoa nos contatos a partir de seu nome (retorna o índice).
    c. Remover uma pessoa dos contatos, a partir de seu nome.
    d. Listar contatos

Práticas

3. Crie uma classe denominada **Elevador** para armazenar as informações de um elevador dentro de um prédio. A classe deve armazenar de forma privada o andar em que se encontra (0=térreo), o total de andares no prédio, a capacidade do elevador (quantas pessoas pode levar), e quantas pessoas estão presentes nele.
4. Crie um construtor para inicializar os dados privados da classe anterior.
5. Incremente a classe Elevador com os seguintes métodos:
    a. entra(): para acrescentar uma pessoa no elevador (se ainda houver espaço);
    b. sai(): para remover uma pessoa do elevador (se houver alguém dentro dele);
    c. sobe(): para subir um andar (se não estiver no último andar);
    d. desce(): para descer um andar (se não estiver no térreo).
    e. consulta_...(): para consultar um dado privado.

Práticas

6. Crie um sistema de gerenciamento de uma biblioteca, definindo as seguintes classes:
    a. Livro: com atributos como título, autor, ISBN e status de disponibilidade.
    b. Usuario: com atributos como nome, ID do usuário e uma lista de livros emprestados.
    c. Biblioteca: com a coleção de livros e de usuários.
7. Na classe Livro, implemente métodos para exibir informações do livro e alterar o status de disponibilidade.
8. Na classe Usuario, implemente métodos para adicionar e remover livros da lista de livros emprestados.
9. Na classe Biblioteca, implemente métodos para adicionar e remover livros, registrar novos usuários, e gerenciar empréstimos e devoluções de livros.