Introdução a Técnicas de Programação

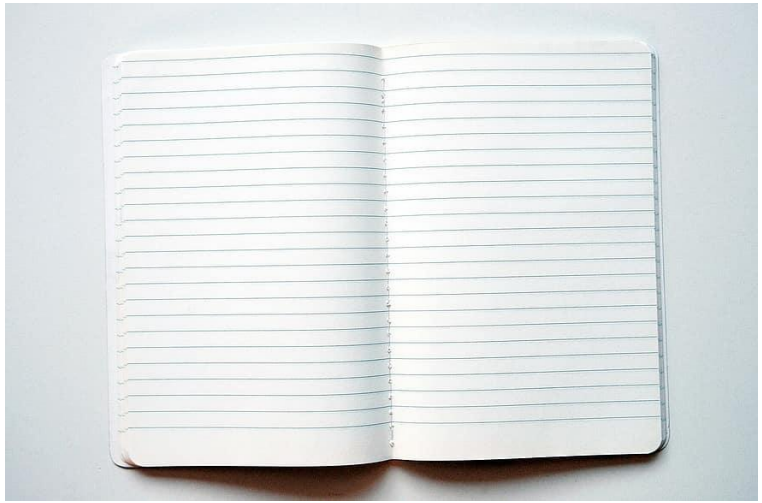
Classes

Prof. André Campos
DIMAp/UFRN

Dos dados estruturados aos objetos

Conceito básico de “objetos”: elementos que possuem

- Dados
- Operações sobre seus dados



Caderno

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

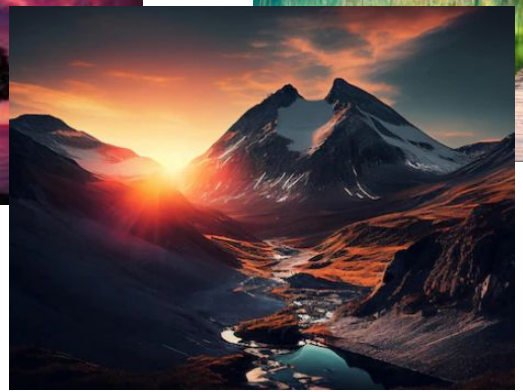
Exemplo

Imagem

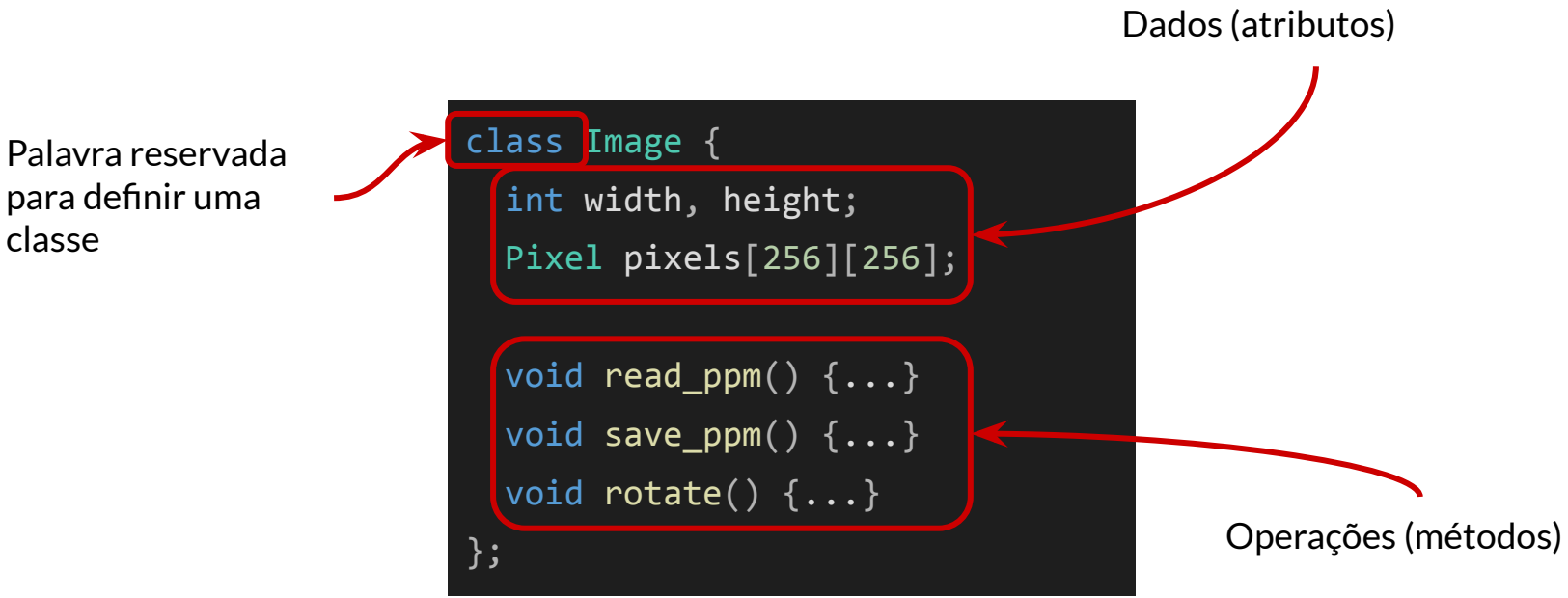
- Dados
  - Largura e altura
  - Matriz de pixels
- Operações sobre os dados
  - Rotacionar
  - Ampliar
  - Mudar escala de cores (ex: cinza)
  - ...

Classes

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



Classes em C++



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...)

```
class Image {
  int width, height;
  Pixel pixels[256][256];

  void read_ppm() {/*...*/}
  void save_ppm() {/*...*/}
  void rotate() {/*...*/}
};

int main() {
  Image original;
  Image escala_de_cinza;
  //...
}
```

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

- Memória
- Circuito integrado
- Fonte 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`**

```
class Image {
  int width, height;
  Pixel pixels[256][256];
public:
  void read_ppm() {/*...*/}
  void save_ppm() {/*...*/}
  void rotate() {/*...*/}
};

int main() {
  Image imagem;
  imagem.read_ppm();
  imagem.rotate();
  imagem.width = 100;  // erro!
  //...
}
```

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.

```
class Image {
  int width, height;
  Pixel pixels[256][256];

public:
  int get_width() { return width; }
  int get_height() { return height; }
  void read_ppm() { /*...*/ }
  //...
};

int main() {
  Image imagem;

  imagem.read_ppm();
  cout << imagem.get_width() << " por ";
  cout << imagem.get_height() << endl;
  //...
}
```

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

```
class Image {
  int width, height;

public:
  Image() { /* ... */ }
  //...
};
```

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.

```
int main() {
  Image imagem;

  //...
}
```

```
class Image {
  int width, height;

public:
  Image() {
    width = 0;
    height = 0;
  }
};
```

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)

```
int main() {
  Image imagem(0, 0);
  //...
}
```

```
class Image {
  int width, height;

public:
  Image(int w, int h) {
    width = w;
    height = h;
  }
};
```

Construtor parametrizado

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

```
int main() {
  Image imagem(0, 0);
  //...
}
```

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

```
class Image {
  int width, height;

public:
  Image(int w, int h): width(w), height(h) {
    // ...
  }
};
```

Construtor default parametrizado

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

```
int main() {
  Image imagem_a;
  Image imagem_b(10, 20);
  //...
}
```

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

```
class Image {
  int width, height;

public:
  Image(int w = 0, int h = 0): width(w), height(h) {
    // ...
  }
};
```

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.

```
int main() {
  Image imagem_a;
  Image imagem_b(imagem_a);
  //...
}
```

```
class Image {
  int width, height;

public:
  Image(const Image& img) {
    width = img.width;
    height = img.height;
  }
};
```