II. Encapsulamento

Private / Public

Construtores / Destrutores

This

Pergunta da Aula

Em relação ao conteúdo interno da classe, como controlar (limitar) o que é visto ou utilizado pelo usuário da classe?

Classe Monitor

```
class Monitor
{
      int resoluçãoX;
      int resolucaoY;
      TipoMonitor tipo;
      string marca;
      bool stLigado;

   public:
      void ligar() { stLigado = true; }
      void desligar() { stligado = false; }
      void alterarResolucao(int x, int y) {...}
};
```

PROBLEMA 1

Classe Monitor

```
class Monitor
{
   ...
   bool stLigado;

   void ligar() {
      stLigado = true;
   }
   void desligar() {
      stligado = false;
   }
   ...
};
```

Classe Monitor

```
class Monitor
{
   ...
   bool stLigado;

   void ligar() {
      stLigado = true;
   }
   void desligar() {
      stligado = false;
   }
   ...
};
```

```
int main() {
   Monitor m1;
   m1.ligar();
}
```

OU

```
int main() {
   Monitor m1;
   m1.stLigado = true;
}
```

Classe Monitor

```
class Monitor
{
   ...
   bool stLigado;
   int contLigado;

   void ligar() {
      stLigado = true;
      contLigado++;
   }
   void desligar() {
      stligado = false;
   }
   ...
};
```

```
int main() {
   Monitor m1;
   m1.ligar();
}
```

OU

```
int main() {
   Monitor m1;
   m1.stLigado = true;
}
```

Como garantir que o usuário da classe utilize o método Ligar??

Visibilidade

- É a forma com que os elementos de uma classe (atributos e métodos) podem ser vistos e utilizados externamente
  - Private
    - somente no interior da classe
  - Protected
    - somente no interior da classe, e de suas herdeiras
  - Public
    - dentro e fora da classe, incluindo usuários através da criação de objetos

Exemplo

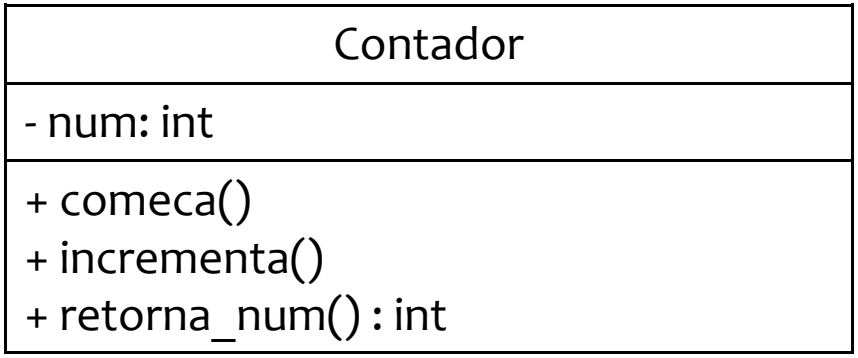
```
class Monitor
{
   private: //implicito
      bool stLigado;
      int contLigado;
      ...
   public:
      void ligar() {
         stLigado = true;
         contLigado++;
      }
      void desligar() {
         stligado = false;
      }
      ...
};
```

Exemplo

```
class Contador
{
   private:
      int num;

   public:
      void inicializa() {num=0;}
      void incrementa() {num=num+1;}
      int retorna_num() {return num;}
};
```

Diagrama UML



Notação UML para acessibilidade

```
+ public
# protected
- private
```

Vantagens do Encapsulamento

- O código torna-se mais claro
  - separa funcionalidades reais das auxiliares
- Minimizam-se os erros de programação
- Interfaces ficam mais simples
- Classes semelhantes podem exibir interfaces semelhantes
- Proporciona facilidades para extensão
- Oculta a realização de modificações
  - alterações que não mexem na interface são transparentes para os usuários da classe

PROBLEMA 2

Classe Monitor – Inicialização??

```
class Monitor
{
   private:
      bool stLigado;
      int contLigado;
      ...
   public:
      void ligar() {
         stLigado = true;
         contLigado++;
      }
      void desligar() {
         stligado = false;
      }
      ...
};
```

contLigado = 0;
stLigado = false;

Como garantir que os dados do objeto sejam inicializados corretamente??

Construtor

- Construtores são funções membro especiais chamadas pelo sistema no momento da criação de um objeto.
- Elas não possuem valor de retorno, porque você não pode chamar um construtor para um objeto (não pode ser chamado explicitamente).
- Construtores representam uma oportunidade de inicializar de forma organizada os objetos.
- Não são obrigatórios.
- Podem ser vazios.
- “Devem” ser públicos

Construtor

- Sintaxe
  - Um construtor tem sempre o mesmo nome da classe.

- Exemplo:
  - Para uma classe string o construtor teria a forma string(char* a); com o argumento char* especificado pelo programador. Ele seria chamado automaticamente no momento da criação, declaração de uma string:
  - string a(“Texto”);
    - //alocação estática implica na chamada do construtor
  - a.mostra();
    - //chamada de métodos estática.

Exemplo

```
class Monitor
{
   private:
      bool stLigado;
      int contLigado;
      ...
   public:
      Monitor() {
         stLigado = false;
         contLigado = 0;
      }
      ...
};
```

Exemplo

```
class Contador
{
      int num;

   public:
      Contador() {num=0;}
      void incrementa() {num=num+1;}
      int retorna_num() {return num;}
};
```

Destrutor

- Nomenclatura
  - idêntica ao construtor
  - com o sinal de “~” na frente
- Não é obrigatório

Exemplo

```
class Contador
{
      int num;

   public:
      Contador() {num=0;}
      ~Contador() {cout<<“objeto destruido”;}
      void incrementa() {num=num+1;}
      int retorna_num() {return num;}
};
```

PROBLEMA 3

Problema 3: Nomes de atributos

```
class Monitor
{
        ...
    void alterarResolucao(int x, int y)
    {
        resX = x;
        resolucaoY = y;
    }
    ...
};
```

Problema 3: Nomes de atributos

```
class Monitor
{
   int resoluçãoX;
   int resolucaoY;
   ...
   void alterarResolucao(int x, int y)
   {
      resolucaoX = x;
      resolucaoY = y;
   }
   ...
};
```

Problema 3: Nomes de atributos

```
class Monitor
{
    int resX;
    int resY;
    ...
    void alterarResolucao(int resX, int resY)
    {
        resX = resX;
        resY = resY;
    }
    ...
};
```

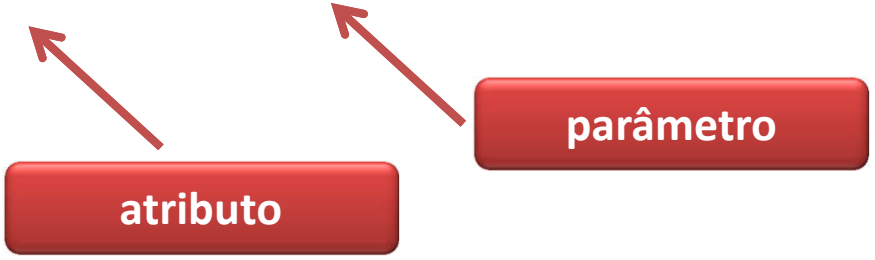
E se eu quisesse que parâmetros e atributos tivessem o mesmo nome??

THIS

- This é uma palavra reservada e pode ser usada dentro de qualquer função membro
- This é um ponteiro para o próprio objeto em questão.
- This **garante** o acesso a um atributo do objeto.

Problema 3: Nomes de atributos

```
class Monitor
{
    int resX;
    int resY;
    ...
    void alterarResolucao(int resX, int resY)
    {
        this->resX = resX;
        this->resY = resY;
    }
    ...
};
```



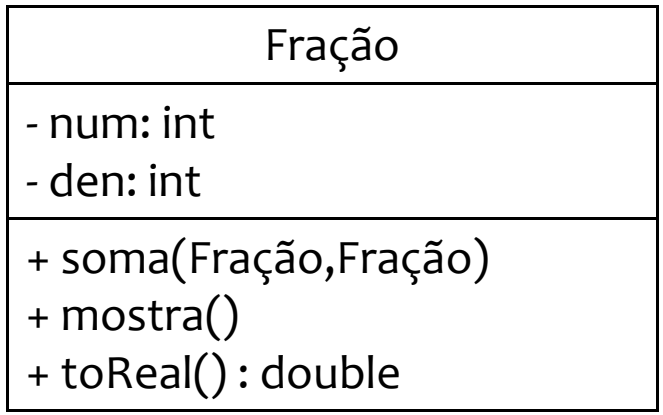
Exemplo

```
class Contador
{
      int num;
    public:
      Contador() {this->num=0;}
      void incrementa() {num=num+1;}
      int getNum() {return num;}
};
```

tanto faz o uso do this neste caso

EXERCÍCIOS

Programe a Classe Fração



Exercícios

- Encapsular as seguintes classes:
  - Caixa
  - Fração
- Criar construtores
- E usar o this quando necessário